

Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

**Sistema de Aquisição de Dados em Microcontrolador e
Comunicação pelo Protocolo Modbus**

Autor:

Raphael Fernandes Vilela

Orientador:

Ulisses de Araújo Miranda, M. Sc.

Coorientador:

Prof. Carlos Fernando Teodósio Soares, D. Sc.

Examinador:

Prof. Gelson Vieira Mendonça, Ph.D.

Examinador:

Prof. Luís Guilherme Barbosa Rolim, Dr.-Ing.

DEL

Agosto de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica - Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro - RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmear ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do autor e dos orientadores.

DEDICATÓRIA

A minha mãe e minha avó, pela paciência com as noites de luzes acesas. Aos amigos Luciano Leite e Rafael Mendes, pelo apoio moral e incentivo.

AGRADECIMENTO

Agradeço a todos os colegas da Recriar Tecnologias pelo suporte técnico e simpatia, em especial ao Ulisses, que esteve disponível para tirar dúvidas.

RESUMO

Este trabalho descreve o desenvolvimento de um equipamento para aquisição de valores de tensão utilizando um microcontrolador ARM. Este equipamento será parte de um sistema de supervisão e monitoramento de dados (SCADA). Os dados obtidos pelo conversor analógico-digital do ARM são enviados para o serviço SCADA por meio de comunicação serial, utilizando o protocolo Modbus.

Palavras-Chave: modbus, protocolos de comunicação, modbus rtu, modbus ascii, modbus, microcontroladores, LPC, ARM, RS232, eletrônica embarcada

ABSTRACT

This work shows the procedure to acquire voltage data with an ARM microcontroller, using the modbus protocol. The voltage is read by the ARM's Analog-Digital converter via serial connection and shown with a Supervisory Control and Data Acquisition (SCADA) interface.

Key-words: modbus, Communications protocols, modbus rtu, modbus ascii, microcontrollers, LPC, ARM, RS232

SIGLAS

A/D - Conversor Analógico Digital

ARM - *Advanced RISC Machine*

Ascii - *American Standard Code for Information Interchange*

CRC - *Cyclic Redundancy Check*, Verificação Cíclica de Redundância

DC - *Direct Current*, corrente contínua

GPIO - *General Purpose In/Out*

IAP - *In-Application Programming*, Programação durante o acesso da aplicação

IDE - *Integrated Development Environment*

IHM - Interface Homem-Máquina

I/O - *In/Out*, terminais de entrada ou saída digital.

JTAG - *Joint Test Action Group*

LRC - *Longitudinal Redundancy Check*, Verificação Longitudinal de Redundância

RTU - *Remote Terminal Unit*

Rx - Receptor

SCADA - *Supervisory Control and Data Acquisition*, Sistemas de Supervisão e Aquisição de Dados

Tx - Transmissor

UART - *Universal Asynchronous Receiver/Transmitter*

XOR - *Exclusive OR*, Ou-Exclusivo (operação lógica)

São usadas as notações 0x0N (onde N é 0,1,...,9,A,...F) para um número hexadecimal e \$(0N) para o caracter cujo valor na tabela Ascii for 0x0N. Esta última notação é de um dos programas utilizados, o Hercules SETUP utility.

Sumário

1	Introdução	1
1.1	Tema	1
1.2	Delimitação	1
1.3	Justificativa	2
1.4	Objetivos	2
1.5	Metodologia	3
1.6	Descrição	4
2	Motivação	6
2.1	Protocolos Industriais	6
2.2	Protocolo Modbus	6
2.3	Microcontroladores	7
2.4	Arquitetura ARM	7
2.5	Objetivo	8
3	Metodologia	9
3.1	Montagem da Rede	9
3.2	Conversão A/D para medição	10
3.3	Material	12
3.4	<i>Softwares</i>	13
4	Protocolo Modbus	16
4.1	Panorama Geral	16
4.2	Protocolo Modbus Ascii	17
4.3	Protocolo Modbus RTU	21

5	Controle de caracteres RTU	24
5.1	Ajuste do <i>clock</i>	24
5.2	Contagem do tempo	26
5.3	Comparação de tempo	27
5.4	Resumo dos registradores	27
5.4.1	<i>Prescale Counter</i> - PC	27
5.4.2	<i>Prescale Register</i> - PR	27
5.4.3	<i>Time Counter</i> - TC	28
5.4.4	<i>Time Counter Register</i> - TCR	28
5.4.5	<i>Match Register</i> - MR	28
5.4.6	<i>Match Control Register</i> - MCR	28
5.5	Programação	29
5.6	Resultados	31
6	Funções Modbus	33
6.1	Formato Geral	33
6.1.1	Função 0x02 - <i>Read Input Status</i>	33
6.1.2	Função 0x01 - <i>Read Coil Status</i>	36
6.1.3	Função 0x04 - <i>Read Input Registers</i>	36
6.1.4	Função 0x03 - <i>Read Holding Registers</i>	37
6.1.5	Função 0x05 - <i>Force Single Coil</i>	38
6.1.6	Função 0x06 - <i>Preset Single Register</i>	40
6.1.7	Função 0x0F - <i>Force Multiple Coils</i>	40
6.1.8	Função 0x10 - <i>Preset Multiple Registers</i>	41
6.1.9	Erro	42
6.2	Implementação	43
6.2.1	Percorrer o número de <i>bytes</i>	43
6.2.2	Ler I/O	46
6.2.3	Escrever no <i>display</i>	46
6.2.4	Cálculo do LRC	49
6.2.5	Cálculo do CRC	49
6.2.6	Resposta sem erro	49
6.2.7	Resposta com erro	51

7	Realização do experimento	52
7.1	SCADA	52
7.2	Uso do ScadaBR	53
7.3	Resultados	54
7.3.1	Ajuste do SCADA usando parâmetros do circuito	56
7.3.2	Ajuste do SCADA calibrando os resultados	57
8	Conclusão	60
	Bibliografia	62
A	Tabela Ascii	64
B	Fluxograma dos registradores de tempo	67

Lista de Figuras

1.1	Divisão da mensagem Modbus	4
3.1	Conexão serial entre um computador e o ARM	9
3.2	Circuito de entrada do A/D	10
3.3	Resposta em frequência (em Hz) do circuito de entrada do A/D . . .	12
3.4	Adaptador LPC x USB (LPC-Link)	13
3.5	IHM	13
3.6	Interface do LPCXpresso	14
3.7	Informações sobre o LPCXpresso	15
3.8	Hercules SETUP utility	15
4.1	Rede Modbus típica	16
4.2	Cálculo do LRC em uma planilha, passo a passo	19
4.3	Montagem do pacote Modbus Ascii	20
5.1	Comportamento dos diferentes registradores	27
5.2	Fluxograma resumido do controle de início e fim de mensagem	31
6.1	Botões do protótipo e os endereços definidos	34
6.2	Endereços das posições de caracteres do <i>display</i>	40
7.1	Diagrama de blocos do sistema conectado à <i>internet</i>	52
7.2	Criação de um <i>data source</i>	54
7.3	Criação de um <i>data point</i>	55
7.4	Acompanhamento em tempo real de uma variável discreta	55
7.5	Escrita de variáveis discretas e acendimento de LED	56
7.6	Escrita de valores contínuos	56
7.7	Reta de calibração dos valores digitais	59

7.8	Gráfico de valores de tensão (em Volts) mostrados pelo ScadaBR . . .	59
B.1	Fluxograma indicando os registradores de tempo	67

Lista de Tabelas

3.1	Medidas de tensão contínua para o circuito de ponderação do A/D . .	11
4.1	Formato geral de uma mensagem Modbus	17
4.2	Exemplo de uma mensagem Modbus Ascii	17
4.3	Exemplo de uma mensagem Modbus RTU	21
5.1	Registrador de <i>clock</i> principal - MAINCLKSEL	25
5.2	Cálculo do fator P, para o PLL	26
5.3	Registrador contagem de tempo - TCR	28
5.4	<i>Match Controller Register</i> - MCR	28
5.5	Comparação entre os tempos de silêncio esperado e obtido	32
6.1	Campo de dados da função 0x01 na solicitação	34
6.2	Campo de dados da função 0x01 na resposta	35
6.3	Exemplo de estado dos terminais de I/O	35
6.4	Exemplos do uso da função 0x01	36
6.5	Exemplos do uso da função 0x02	37
6.6	Campo de dados da função 0x03 na resposta	37
6.7	Exemplo do uso da função 0x03	37
6.8	Campo de dados da função 0x05 na pergunta	38
6.9	Exemplos do uso da função 0x05	39
6.10	Campo de dados do pedido na função 0x06	40
6.11	Exemplos do uso da função 0x06	41
6.12	Campo de dados da função 0x0F na solicitação	41
6.13	Campo de dados da função 0x0F na resposta	42
6.14	Exemplos do uso da função 0x0F	42
6.15	Campo de dados da função 0x10 na solicitação	43

6.16	Campo de dados da função 0x10 na resposta	43
6.17	Exemplos do uso da função 0x10	44
6.18	Fontes de erro Modbus	44
6.19	Mensagem Modbus e número de caracteres por bloco	45
6.20	Exemplos de respostas a erros	45
6.21	Mapa I/O	46
7.1	Valores digitais obtidos no ScadaBR	57
7.2	Valores medidos usando parâmetros do circuito	58
7.3	Valores medidos usando o método dos mínimos quadrados	58

Capítulo 1

Introdução

1.1 Tema

Existem produtos no mercado disponíveis para comunicar dados utilizando protocolos de comunicação industriais. Neste projeto, foi desenvolvido um produto com este propósito, utilizando um microcontrolador para obter medidas de tensão ou corrente e enviá-las para um computador conectado por um cabo serial

A programação do microcontrolador, em linguagem C, envolve o conhecimento da “unidade básica de dados” - padrão de *bits* que caracteriza o protocolo - e do desenvolvimento das funções de leitura e escrita deste protocolo.

1.2 Delimitação

Os equipamentos e protocolos escolhidos são amplamente difundidos e testados, e suas limitações são tais que permitem a execução do projeto a ser realizado. Entretanto, existem outras interfaces SCADA (Sistemas de Supervisão e Aquisição de Dados) e protocolos que não os usados neste projeto e estes não serão aqui avaliados. Também não será avaliada a arquitetura de funcionamento do microcontrolador. Vale, por fim, ressaltar que os resultados obtidos pelo produto desenvolvido não possuem precisão suficiente para uso em, por exemplo, questionamentos de tarifas cobradas em contas de luz.

1.3 Justificativa

O monitoramento à distância é uma prática muito importante para a indústria, envolvendo sensoreamento, microcontroladores, transmissão de dados e interfaceamento com o usuário final. Para adquirir os dados dos sensores, pode ser utilizado um microcontrolador com núcleo de ARM (sigla para *Advanced RISC Machine*), dispondo de um conversor Analógico/Digital (A/D) e um conector Rx/Tx (Receptor/Transmissor). A arquitetura ARM é encontrada na grande maioria dos núcleos de aparelhos móveis [1], garantindo segurança e confiabilidade, com grande performance, códigos mais enxutos e baixo consumo de energia [2]. Além disso, com o programa LPCXpresso, a programação se torna prática - em linguagem C.

Para a interface visual, um programa do tipo SCADA facilita a comunicação, ao ponto em que monta os chamados pacotes de mensagem automaticamente, se adequando ao protocolo utilizado e à porta de comunicação e seus parâmetros, permitindo que a comunicação seja bem sucedida, desde que a programação daqueles com quem está se comunicando esteja correta. O protocolo de comunicação utilizado é o Modbus, devido à sua ampla utilização na indústria, de forma aberta [3], gratuita, de fácil manutenção e bem padronizada.

Vale ressaltar que não foi encontrada nenhuma biblioteca aberta para implementar o protocolo Modbus, fazendo com que o desenvolvimento do protocolo durante o projeto agregue valor ao desenvolvimento futuro de equipamentos usando o protocolo Modbus.

1.4 Objetivos

Desenvolver um equipamento de medição de tensão e corrente, utilizando protocolo Modbus através de um microcontrolador ARM, o qual atuará como escravo, recebendo requisições de um software SCADA. O trabalho desenvolvido consiste em programar o ARM para reconhecer as mensagens recebidas e realizar as funções desejadas.

1.5 Metodologia

A realização de medidas exige, como primeiro passo, uma comunicação local via RS-232 (porta serial) entre o computador e o microcontrolador LPC1113. Para desenvolver esta comunicação, é preciso definir um endereço para o LPC, pois este atuará como escravo. Sua programação é feita através de um conector entre a porta USB do computador e o dispositivo propriamente dito e utiliza o programa LPCXpresso, baseado na conhecida IDE (*Integrated Development Environment*, ou ambiente de desenvolvimento) Eclipse, o qual possui algumas funções base para a programação de certos parâmetros do dispositivo, como frequência de *clock* e pinagem de entrada e saída. Esta programação de nível mais baixo é feita modificando um arquivo de configuração do modelo de LPC utilizado. Já a programação em um nível mais alto, consistente na comunicação dos *bits* e verificação de erros, é realizada através de funções desenvolvidas durante o projeto.

O protocolo Modbus consiste em mensagens com códigos em hexadecimal possuindo quatro campos, resumidamente: endereço do dispositivo, função, dados e verificação de erro. Por exemplo, suponha a mensagem 110500ACFF004E8B; 11 é o endereço do dispositivo em hexadecimal, 05 é a função equivalente a forçar um valor para uma saída digital, 00ACFF00 são dados sobre esta saída digital e 4E8B é a verificação de erro, formando o chamado pacote de mensagem, que deve ser reconhecido pelo microcontrolador. Para um protocolo Modbus, este pacote pode ser de dois tipos: RTU (sigla para *Remote Terminal Unit*) ou Ascii (*American Standard Code for Information Interchange*, isto é, um padrão para troca de informação). A diferença está no modo como os códigos são enviados: enquanto no Ascii, os *bytes* enviados equivalem ao número na tabela Ascii de cada caracter, no RTU, enviam-se, diretamente, os valores a cada *byte*. Desta maneira, o modo RTU é mais econômico, porém, é importante também desenvolver usando Ascii para garantir compatibilidade com aparelhos mais antigos, em especial Modems.

Finalmente, a interface com o usuário é um programa chamado ScadaBR, onde se deve criar uma *data source*, configurando os padrões da comunicação serial (parâmetros como porta COM e velocidade) e, em seguida, *data points* equivalentes aos parâmetros

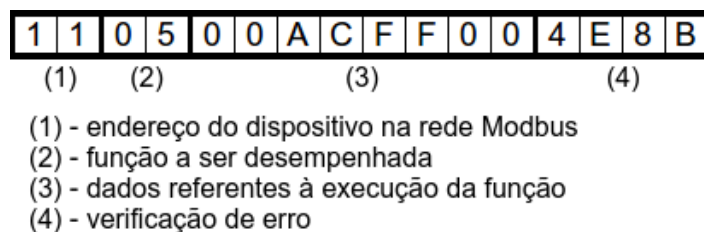


Figura 1.1: Divisão da mensagem Modbus

a serem medidos. Existem locais nesta interface onde é possível ver gráficos da evolução do parâmetro, configurar imagens para indicação de status digital ou analógico e acessar os dados através do servidor Apache Tomcat.

1.6 Descrição

O **Capítulo 2** apresentará um panorama geral dos protocolos industriais, sua importância e a demanda, culminando na apresentação do protocolo Modbus como principal protocolo para comunicação industrial. Há também uma visão geral sobre microcontroladores, focando no caso específico da arquitetura ARM, que processa as informações de comunicação do projeto.

O **Capítulo 3** descreve as fontes de informação consultadas neste projeto e o material de suporte, além dos programas utilizados. Também apresenta como serão feitas as medidas de tensão, mostrando o circuito de entrada do A/D.

Aprofundando o escopo do Capítulo 2, o **Capítulo 4** analisa, com maiores detalhes, a comunicação Modbus, apresentando as estruturas de programação responsáveis para que o microcontrolador possa reconhecer e responder adequadamente a mensagens enviadas por um dispositivo mestre. A estrutura de *bytes* dos pedidos e respostas também são detalhados, incluindo-se como são realizadas as verificações de erro, mostrando as diferenças entre o Modbus Ascii e o Modbus RTU. Na programação do controle de caracteres para mensagens Modbus RTU, verifica-se a necessidade de lidar com controle de tempo.

O **Capítulo 5** mostra como o ARM realiza contagem de tempo e as funções desenvolvidas para atender às demandas da comunicação Modbus com relação à cronometragem.

Serão vistas no **Capítulo 6** as funções Modbus: através do reconhecimento da mensagem enviada, o microcontrolador é programado para fazer leituras ou escritas em pinos e registradores, que podem ter seu estado mostrado para o operador em forma de acendimento de LEDs e mensagens em *display*.

O **Capítulo 7** mostra como um *software* SCADA auxilia o monitoramento de um sistema real e, por meio de mensagens Modbus, exibe informações sobre o sistema de forma amigável. A partir daí, são vistos os resultados do estado de I/O (entrada e saída) e da medida de tensão do conversor A/D.

Na conclusão, resumem-se todas as informações de forma integrada e são sugeridos usos do produto e melhorias, como, por exemplo, monitoramento por meio de *smartphones*.

Capítulo 2

Motivação

2.1 Protocolos Industriais

Quando existe uma forma específica de envio de *bits*, de maneira padronizada, entre dois dispositivos digitais, esta padronização é chamada de protocolo. Protocolos usam unidades básicas de dados, que representam o formato no qual os *bits* são enviados, e, entre eles, os mais famosos são os protocolos TCP e IP, responsáveis pela comunicação da *internet*.

A indústria, através da automação, demanda protocolos para a comunicação entre os diversos equipamentos, conhecidos como protocolos de chão de fábrica, cujas características são o determinismo da rede e menor atraso nas respostas [4]. Existem vários protocolos com este propósito, entre eles, podem ser citados o Profibus, DeviceNet, EtherNet/IP e, principalmente o Modbus, o qual possui a maior documentação aberta dentre os citados.

2.2 Protocolo Modbus

O protocolo Modbus se destaca por ser um padrão internacional organizado e ter alta compatibilidade com equipamentos, tornando-o muito utilizado por processos de automação. Há uma empresa responsável por sua normalização e apoio a protocolos variantes, a *Modbus Organization*, que possui uma página na *internet*[3] com informações para o usuário deste protocolo, como especificações e divulgação

de empresas que desenvolvem equipamentos compatíveis.

Sua importância foi conquistada por seu caráter aberto, praticidade e baixo custo e consolidou-se no mercado por meio do Modbus TCP, que expandiu a comunicação via *internet*. Para redes locais, o meio físico pode ser um cabo com conectores RS-232 ou RS-485, e existem dois tipos: Modbus RTU e Modbus Ascii. Já para um sistema embarcado, o acesso remoto é feito pelo Modbus TCP, o qual usa o campo de dados de uma mensagem TCP para enviar a mensagem Modbus RTU e seu meio físico podem ser conexões Ethernet. A troca de mensagens é do tipo Mestre-Escravo, isto é, o dispositivo mestre pede uma informação e o escravo processa e responde, e os *bytes* podem ser enviados como seus valores correspondentes na tabela Ascii ou ter seus valores passados diretamente.

2.3 Microcontroladores

Uma comunicação Modbus pode ser feita entre uma grande variedade de dispositivos. Por exemplo, pode realizar a comunicação entre um computador e um sensor ou microcontrolador, em que o computador realiza a interface e monta as mensagens de pedido, devendo, portanto, o microcontrolador interpretar a mensagem recebida, processar as informações (modificando tensões ou lendo dados, por exemplo) e montar o pacote de resposta. Microcontroladores realizam o processamento de tarefas e contam com pinos periféricos que permitem alterar o estado de alguns sinais elétricos, assim como medi-los. As principais arquiteturas dos microcontroladores são AVR, ARM e PIC, para as quais as diferenças se encontram, principalmente, nas diferentes funcionalidades permitidas (entrada e saída, memórias RAM e *flash*, entre outras), geração de interrupções, consumo de energia e quantidade de *bits*.

2.4 Arquitetura ARM

A escolha pela arquitetura ARM se dá pois possui baixo custo e bom poder de processamento, demandados pelos protocolos de chão de fábrica. A programação é feita em linguagem C, o que é permitido pelo programa LPCXpresso, que possui uma interface baseada em Eclipse e um compilador GNU C [5]. Além disso, a fabricante

vende *kits* de desenvolvimento que facilitam a integração entre o programa e o *chip* em si. Através desses *kits* e de exemplos, também disponíveis pela fabricante, é possível programar em alto nível, requerendo um nível mais baixo apenas para controle interno do microcontrolador, como frequência de *clock* e alteração de entradas e saídas (botões, *LEDs*, *display* e conversor A/D). Alterando os exemplos desenvolvidos pela fabricante e consultando problemas levantados pela comunidade de usuários (através do fórum knowledgebase.nxp.com) e o manual do *kit*, são obtidas funções modificadas para o uso em problemas específicos.

2.5 Objetivo

O objetivo deste projeto é o desenvolvimento de um protótipo, utilizando comunicação Modbus, para medição de tensão ou corrente na empresa Recriar Tecnologias, parceira do Laboratório de Fontes Alternativas de Energia - LAFAE. A partir do protótipo, pretende-se expandir o uso do protocolo em questão para os equipamentos já em operação.

Capítulo 3

Metodologia

3.1 Montagem da Rede

Primeiramente, foi utilizado um exemplo para o uso da porta UART (*Universal Asynchronous Receiver/Transmitter*) do microcontrolador, que serve para fazer comunicação Rx/Tx, sendo composta por um pino de terra, um para recebimento e outro para transmissão de dados. O teste é realizado com a porta UART conectando o microcontrolador ao computador através de um conversor USB-RS232, que simula uma porta serial como de computadores mais antigos, na porta USB, usando um programa de envios de mensagens via serial (como o *hyperterminal* do Windows ou o Minicom do Linux), com a velocidade *baudrate* (taxa de caracteres enviados por segundo) configurada para ser a mesma no computador e no ARM. Para verificar que o que está sendo enviado pelo computador é corretamente recebido no ARM, é possível acompanhar a variável que recebe o valor de Rx pela IDE LPCXpresso.

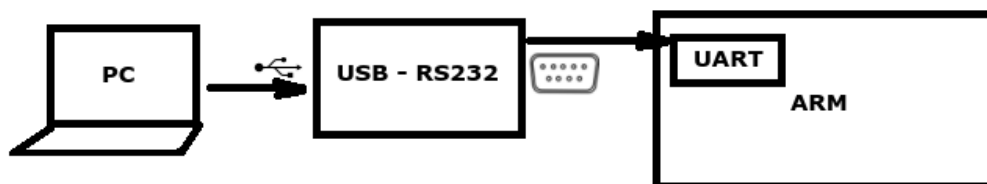


Figura 3.1: Conexão serial entre um computador e o ARM

A partir deste ponto, implementou-se a comunicação Modbus Ascii, usando como mestre um programa instalado no computador que envia pacotes Modbus prontos. A comunicação exige a verificação de dados, portanto, foi necessário, neste ponto, também entender o funcionamento da verificação horizontal de redundância LRC (sigla para *Longitudinal Redundancy Check*) e implementá-la. A implementação da comunicação Modbus RTU é análoga à Ascii, com a diferença de que há a necessidade de realizar uma contagem de tempo, onde foram realizados testes para certificar a validade das frequências de *clock* obtidas e, assim como no Modbus Ascii, estudou-se e implementou-se a verificação de erros, a verificação cíclica de redundância, CRC (da sigla em inglês, *Cyclic Redundancy Check*).

3.2 Conversão A/D para medição

Para realizar a medição, é preciso saber as especificações da tensão de entrada para que seja convertida em valores mensuráveis pelo conversor A/D do ARM, sem riscos de sobretensão no mesmo. Os valores medidos pelo A/D devem estar entre 0 e 3,0 V [6] e a tensão medida varia entre -12 a 12 V, sendo projetado, então, o circuito da figura 3.2.

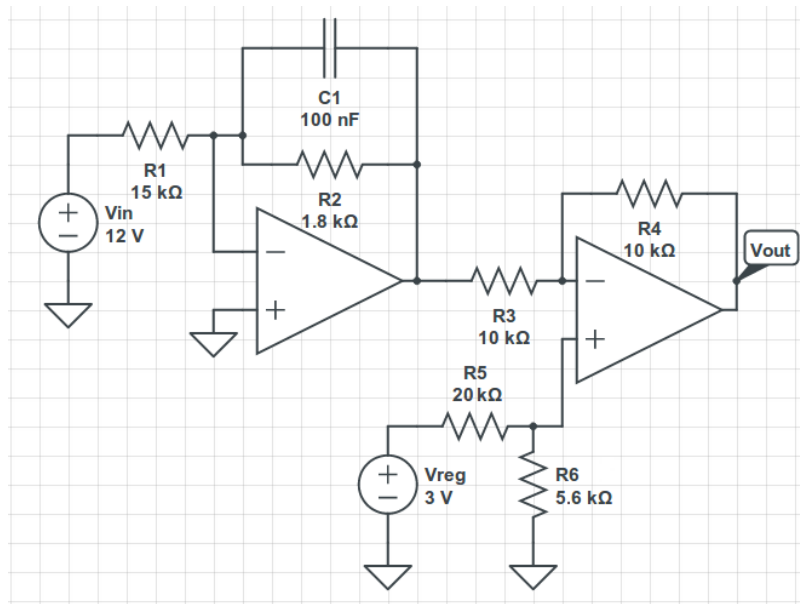


Figura 3.2: Circuito de entrada do A/D

O circuito apresenta a redução de 12 V para 1,4 V e a inclusão de um *offset* de 1,3 V, gerado por um divisor resistivo cuja tensão de alimentação é uma tensão de 3,0 V regulada, dada pelo próprio ARM. Este *offset* serve para que os valores negativos de tensão de entrada sejam mapeados entre 0V e 3,0V. A alimentação do Amplificador Operacional utilizado, o TL072, é fornecida por uma fonte de 15V e -15V, portanto, a tensão de saída do circuito é dada pela equação (3.1).

$$V_{out} = \frac{V_{entrada}}{8,3} + 1,3 \quad (3.1)$$

O A/D do ARM realiza conversão por aproximações sucessivas [6], resultando em 10 *bits*, com valor de referência igual a 3,0V e erro de medida igual a ± 4 . Os valores digitais fornecidos pelo microcontrolador são dados pela equação (3.2), onde V_{out} é o valor da tensão já ponderado pelo circuito da figura 3.2.

$$\text{Valor Digital} = \frac{1023V_{out}}{3,0} \quad (3.2)$$

Os valores simulados e os medidos estão apresentados na tabela 3.1. Além disso, o circuito tem um capacitor responsável por compôr o filtro *Anti-Aliasing*. Deseja-se filtrar frequências de até 60 Hz, então, projetou-se o filtro para frequências de até 100 Hz, para dar margem. A frequência de corte é dada pela equação (3.3).

Tabela 3.1: Medidas de tensão contínua para o circuito de ponderação do A/D

$V_{entrada}$	Saída Esperada	Saída Simulada	Saída Medida
-12V	-0,14V	-0,06V	-0,08V
-6V	0,58V	0,63V	0,62V
0V	1,3V	1,3V	1,3V
6V	2,0V	2,0V	2,1V
12V	2,7V	2,6V	2,7V
16V	3,2V	3,3V	3,3V

$$f_c = \frac{1}{2\pi \times 1,8k\Omega \times 100nF} \approx 884Hz \quad (3.3)$$

Uma simulação do tipo *AC Sweep* foi feita para verificar a atuação do filtro *Anti-Aliasing* e pode ser vista na figura 3.3. Nela, é possível perceber que a faixa de frequências de interesse, de DC a 60 Hz, estão na banda passante (por volta de -18 dB , conforme o projetado, $-20\log 8,3$).

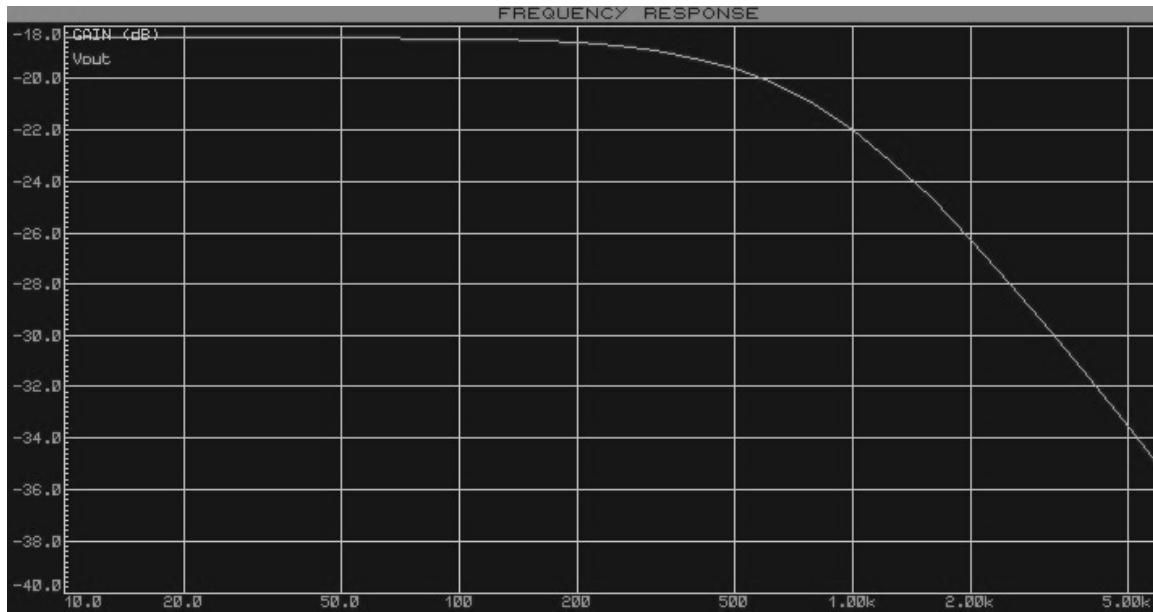


Figura 3.3: Resposta em frequência (em Hz) do circuito de entrada do A/D

3.3 Material

Foi usado o *kit* de desenvolvimento LPC1113, cuja programação é feita através do dispositivo de *debug JTAG* (*Joint Test Action Group*) LPC-Link, como o mostrado na figura 3.4. O *kit* conta com uma interface Homem-Máquina (IHM) com botões, LEDs e *display* diretamente conectados aos terminais de entrada e saída do microcontrolador (figura 3.5). A comunicação serial é feita através de um cabo UART-Serial e de um Serial-USB.

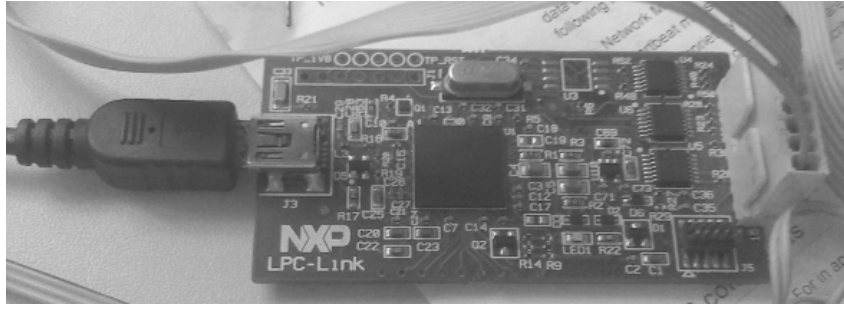


Figura 3.4: Adaptador LPC x USB (LPC-Link)

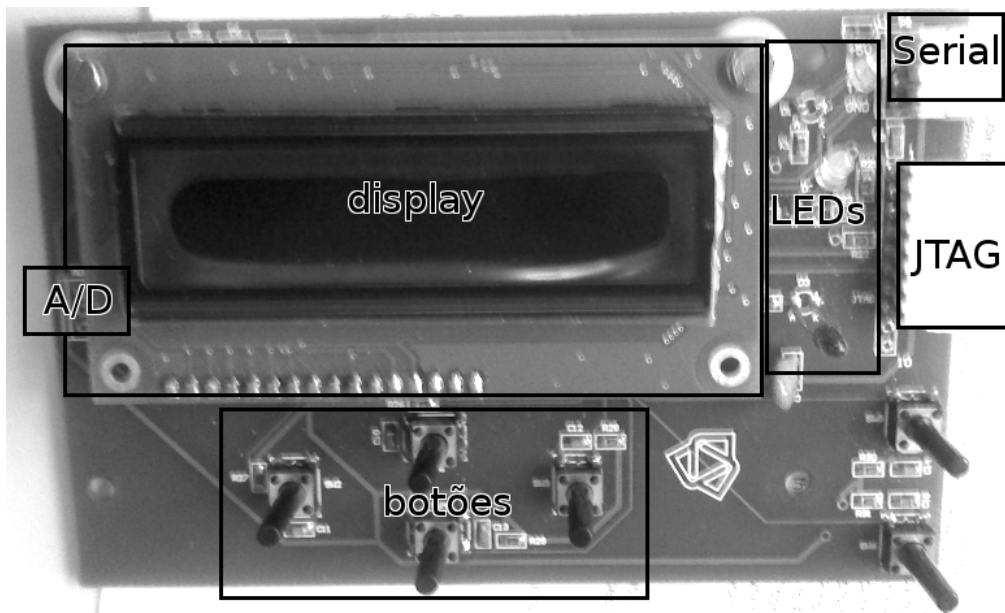


Figura 3.5: IHM

3.4 *Softwares*

Para a comunicação serial, inicialmente, utilizou-se uma versão para testes do programa Modbus Poll, disponível em sua página [7]. O programa envia pedidos já no formato Modbus Ascii ou RTU, realizando a verificação dos dados retornados e dá um retorno visual na forma de uma tabela com valores. Por ser uma versão de testes, após seu período de expiração, foi substituído pelo Hercules SETUP Utility, programa grátis que permite enviar mensagens quaisquer em formato de código hexadecimal ou o caracter ascii diretamente.

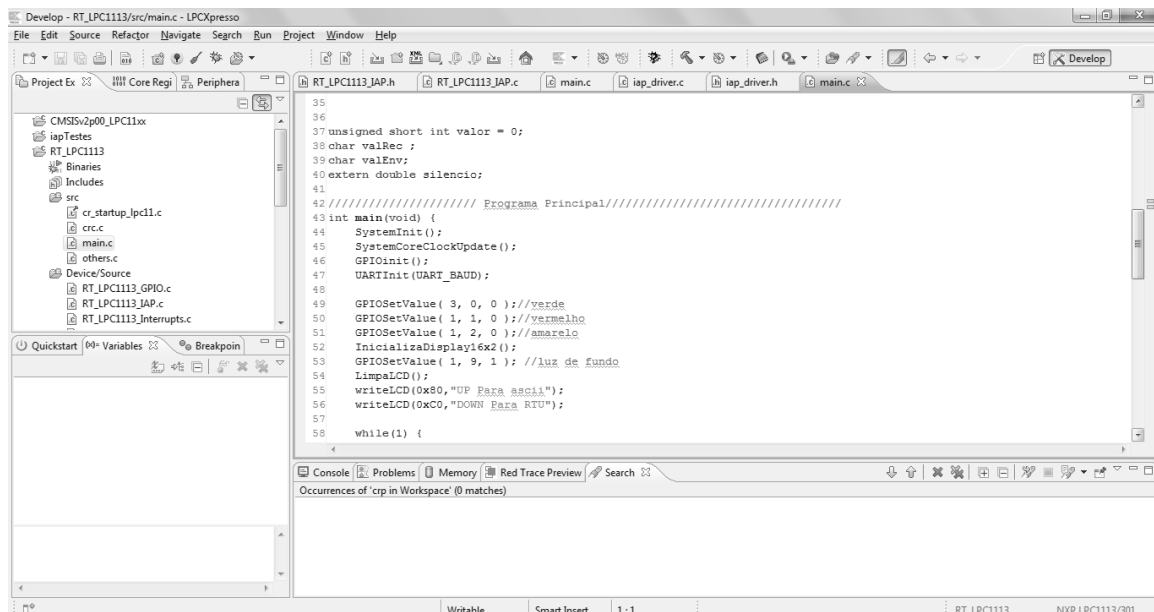


Figura 3.6: Interface do LPCXpresso

O principal programa é o LPCXpresso, de distribuição gratuita com necessidade apenas de um cadastro na página da desenvolvedora, com o qual é possível programar o ARM, através das bibliotecas específicas, e realizar *debug* com o JTAG.

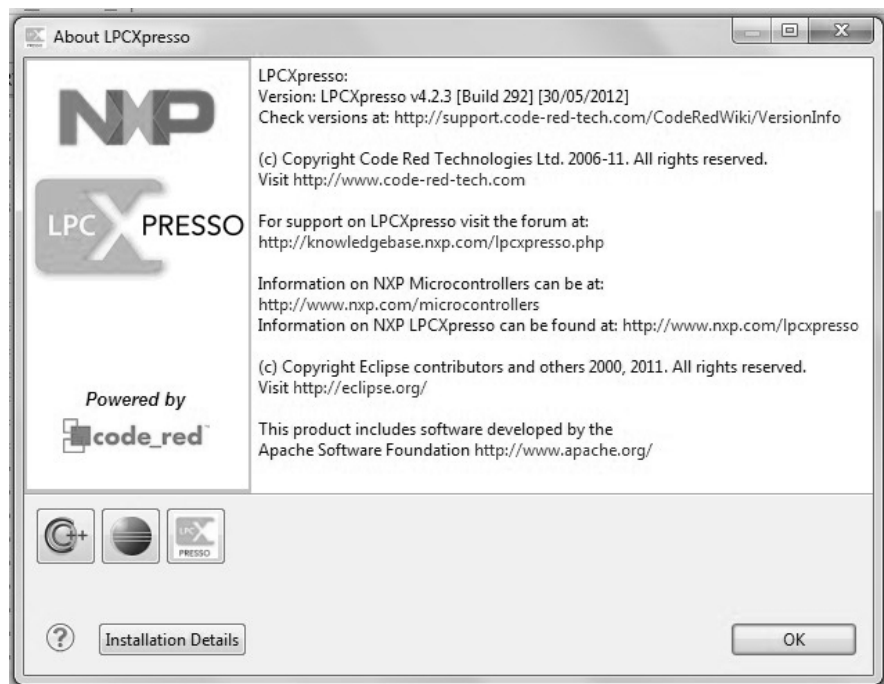


Figura 3.7: Informações sobre o LPCXpresso

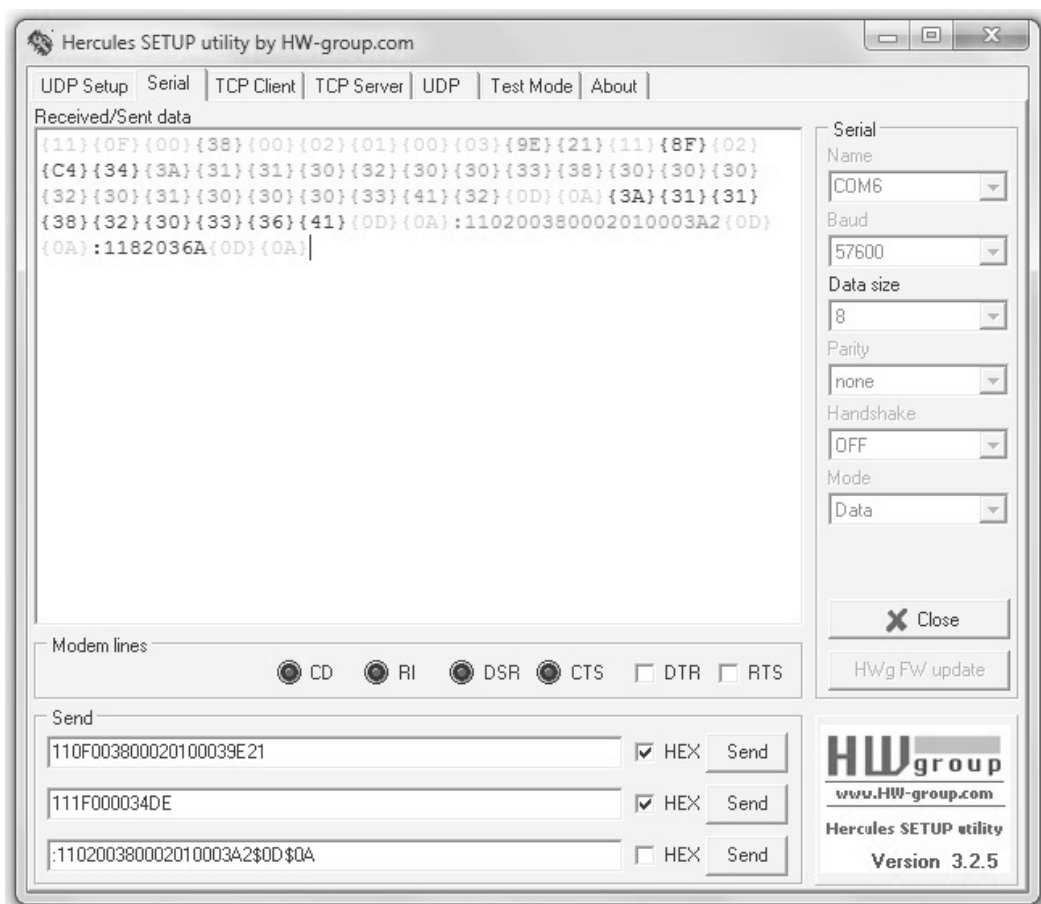


Figura 3.8: Hercules SETUP utility

Capítulo 4

Protocolo Modbus

4.1 Panorama Geral

A rede Modbus conecta o mestre a todos os escravos, enviando uma mensagem ao longo de toda a rede. Portanto, cada escravo deve ter um endereço individual dentro da rede Modbus. Sendo o objetivo do projeto desenvolver o dispositivo escravo da rede, é preciso que o mesmo identifique o endereço. Se a mensagem for endereçada a ele, deve prosseguir com os comandos do protocolo; senão, ele volta ao estado de espera por comando. Uma rede Modbus pode ser ilustrada como na figura 4.1.

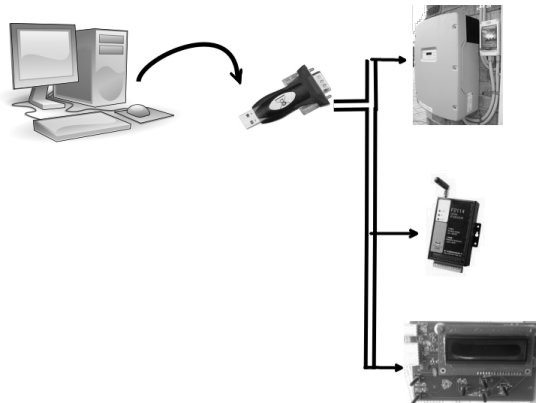


Figura 4.1: Rede Modbus típica

Uma mensagem utilizando o protocolo Modbus pode ser caracterizada em quatro partes principais, mostradas na tabela 4.1.

Tabela 4.1: Formato geral de uma mensagem Modbus

Endereço	Função	Comando	Verificação de erros
----------	--------	---------	----------------------

O primeiro campo, **endereço**, é o que identifica o dispositivo dentro de uma rede Modbus, como a ilustrada na figura 4.1. O campo de **função** ordena ao dispositivo selecionado que realize uma determinada tarefa, uma dentro das quais serão descritas no Capítulo 6. O campo **comando** envia informações complementares para realizar a tarefa pedida no campo anterior e, por fim, o campo de **verificação de erros** é resultado de um cálculo, com base em todos os *bytes* anteriores, que determina a consistência entre o que foi enviado pelo mestre e o que foi recebido pelo escravo. O modo como os caracteres da mensagem são enviados e recebidos divide este protocolo em dois tipos: RTU e Ascii.

4.2 Protocolo Modbus Ascii

Tabela 4.2: Exemplo de uma mensagem Modbus Ascii

Marcador	Endereço	Função	Escopo da Função	Verificação	Marcadores
:	11	05	0034FF00	B7	CR LF

Neste formato, a informação enviada a cada *byte* é o valor, na tabela Ascii, de cada um dos caracteres. Além disso, é caracterizada por conter identificadores de início e fim de mensagem, que são, respectivamente, o caracter ‘:’ (0x3A) e os caracteres ‘\r’ (0x0D, *carriage return*) e ‘\n’ (0x0A, *line feed*). No exemplo acima, os valores ascii de cada caracter da mensagem enviada são 0x3A 0x31 0x31 0x30 0x35 0x30 0x30 0x33 0x34 0x46 0x46 0x30 0x30 0x42 0x37 0x0D 0x0A.

Para que o microcontrolador possa manipular as mensagens Modbus, foi usada a seguinte estrutura de dados:

```
typedef struct {
```



```

    char end0;
    char end1;
    char cmd0;
    char cmd1;
    char dados[TAMANHO.DADOS];
    char lrc0;
    char lrc1;
    int tamanho;
} pacote_ascii;

```

A função do atributo “tamanho” é garantir que somente informações pertinentes sejam carregadas no vetor de dados, além de ser útil para a realização de algumas das funções Modbus.

A verificação de erros realizada é a chamada Verificação Longitudinal de Redundância, ou, em inglês, *Longitudinal Redundancy Check*. O procedimento para verificar a consistência dos dados, a partir de uma sequência de caracteres, segue o seguinte algoritmo: [8]

(1) Os caracteres são somados, aos pares, ou seja, cada termo da soma ocupa 1 byte. Por exemplo, para a mensagem :0104020A11, é feita a soma $0x01 + 0x04 + 0x02 + 0x0A + 0x11 = 0x22$.

(2) Do resultado, obtém-se o complemento de 2, que, na prática, equivale a somar $0xFFFF$ com o negativo do resultado de (1), e, depois, somar $0x01$. Seguindo o exemplo, temos $0xFFFF - 0x22 = 0xFFDD$ e $0xFFDD + 0x01 = 0xFFDE$. Ignorando os possíveis *carries*, é obtido o byte referente ao LRC nos dois últimos algarismos do resultado.

Antes de implementar em linguagem C, o algoritmo foi testado em um programa de cálculos com planilhas, comparando com exemplos de mensagens Modbus Ascii [7]. Para calcular o LRC usando a planilha eletrônica, a operação de soma deve ser feita entre números na base decimal, portanto, os bytes são convertidos para a

base decimal, somados e, então, convertidos novamente para hexadecimal. A figura 4.2 apresenta as etapas do cálculo através da planilha, onde a célula A2 representa a mensagem cujo LRC será calculado, a coluna B separa os *bytes* da mensagem, a coluna C transforma os trechos da mensagem em decimal, a célula D2 apresenta a soma da coluna C, em valor decimal, a célula D5 contém o negativo da soma em hexadecimal e, finalmente, a célula E2 contém o LRC.

f_x	A	B	C	D	E
1	Entrada	bytes	hex2dec	soma	LRC
2	0104020A11	01	1	34	DE
3		04	4		
4		02	2	dec2hex(-soma)	Mensagem
5		0A	10	FFFFFFFDE	:0104020A11DE
6		11	17		
7					
8					

Figura 4.2: Cálculo do LRC em uma planilha, passo a passo

Entendido o algoritmo, partiu-se para a sua implementação em C:

```

unsigned char calculaLRC (pacote_ascii meuPacote){
    unsigned char resultado;
    char i;
    unsigned char cmd = char2hex(meuParamote.cmd0)*0x10+
                        char2hex(meuParamote.cmd1);
    unsigned char end = char2hex(meuParamote.end0)*0x10+
                        char2hex(meuParamote.end1);
    unsigned int soma = cmd+end;
    for (i=0;i<meuParamote.tamanho;i++){
        unsigned char hexa=meuParamote.dados[i];
        hexa=char2hex(hexa);
        if (i%2==0) hexa=hexa*0x10;
        soma+=hexa;
    }
    soma=0xFFFF-soma+1;
}

```

```

    resultado=soma%0x100;
    return resultado;
}

```

O fluxograma resumido da figura 4.3 mostra como é feita a montagem do pacote Modbus Ascii.

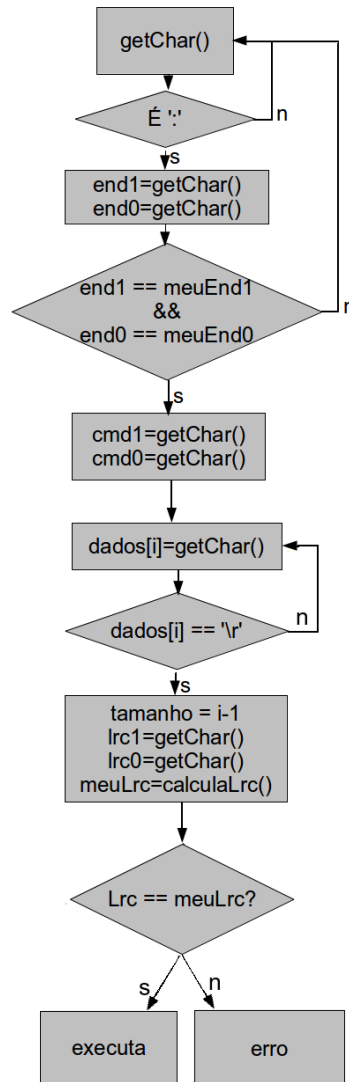


Figura 4.3: Montagem do pacote Modbus Ascii

Tabela 4.3: Exemplo de uma mensagem Modbus RTU

Endereço	Função	Escopo da Função	Verificação
11	05	0034FF00	CF64

4.3 Protocolo Modbus RTU

Nas mensagens RTU, é possível notar a ausência de marcadores de início ou fim de mensagem, substituídos por “silêncios”, isto é, se não for detectada a chegada de caracteres pelo equivalente em tempo de 3,5 caracteres [9], o sistema deve considerar que o próximo caracter é uma potencial mensagem, caso contrário, deve ignorar, pois é uma mensagem para outro sistema, o qual ainda não terminou a comunicação. O tempo de envio de um caracter é caracterizado pela quantidade de *bits* por símbolo, ou seja, com quantos *bits*, incluindo *bits* de verificação de erro, e o tempo de envio do símbolo, a *baudrate*.

Da mesma maneira, o sistema deve receber os caracteres (inserindo-os em uma estrutura de dados semelhante à que foi apresentada para o pacote de dados Ascii) até que não receba mais nada após o mesmo intervalo de tempo. Para isto, utilizou-se um sinal de *clock* através de um cristal presente no microcontrolador e, definida *a priori* o *baudrate*, ou seja, a quantidade de caracteres por unidade de tempo, de maneira que o tempo de espera é:

$$T = \frac{3,5 \times n_{bits}}{Baudrate} \quad (4.1)$$

Cada *byte* da mensagem é, diretamente o valor a ser trocado, sendo, por este motivo, mais econômico que o Ascii, pois, a cada *byte*, são enviados dois caracteres. Assim, a estrutura de dados fica modificada como mostrado a seguir:

```
typedef struct {
    unsigned char end;
    unsigned char cmd;
    unsigned char dados [TAMMAX];
    unsigned int tamanho;
```

```

        unsigned short int crc;
    } pacote_rtu;

```

Para o Modbus RTU, a verificação de dados é a CRC16 (16 *bits*) [10], que consiste em uma verificação vertical de paridade feita ao longo de toda a mensagem. O algoritmo atribui um valor inicial de 0xFFFF para o CRC, e, a cada byte (par de caracteres), faz as seguintes operações: [11]

1) Realiza-se uma operação XOR entre o último resultado para o CRC, (inicialmente, 0xFFFF) e o byte.

2) O bit menos significativo é analisado: se for igual a 0, o resultado sofre um deslocamento para a direção menos significativa, preenchendo-se com um 0 na posição mais significativa; se for igual a 1, além desta operação de deslocamento, faz-se outro XOR entre o resultado parcial e um valor determinado (para o CRC16, este valor é 0xA001 [12]). Esta operação se repete até percorrer todos os bits do par de caracteres atual.

3) Retorna à operação 1 para o próximo byte até encontrar o fim da mensagem.

A implementação, em C, deste algoritmo, é feita pelas funções abaixo:

```

unsigned short  calculaCRC(pacote_rtu pacote){
    unsigned short  resultado=0xFFFF;
    unsigned char  operando;
    unsigned short  i;

    operando=pacote.end;
    resultado=resultado ^ operando;
    resultado=loopInterno(resultado);
    operando=pacote.cmd;
    resultado=resultado ^ operando; //XOR
    resultado=loopInterno(resultado);
    unsigned char  buf[20];

```

```

for ( i=0; i<(pacote.tamanho); i++){
    resultado=resultado ^ pacote.dados[i];
    resultado=loopInterno(resultado);
}
resultado=(resultado%0x100)*0x100+(resultado/0x100);
return resultado;
}

unsigned short loopInterno(unsigned short resultadoParcial){
    unsigned char i,operando;
    unsigned short int resultado=resultadoParcial;
    for ( i=0; i!=8; i++){
        operando=resultado%2;
        resultado=resultado>>1;
        if (operando==1)    resultado=resultado ^ VALOR;
    }
    return resultado;
}

```

Capítulo 5

Controle de caracteres RTU

Como foi mencionado no Capítulo 4, a troca de mensagens RTU utiliza como marcadores a ausência de caracteres por um período equivalente a 3,5 caracteres. Para obter o tempo da equação (4.1), é necessário alterar registradores de *timer*, cujos pinos são mapeados pelo arquivo LPC11xx.h, presente nas configurações padrão do LPCXpresso.

5.1 Ajuste do *clock*

No ARM, há uma série de registradores que, conforme o valor carregado, configuram outros registradores para fazer contagem proporcional ao período do pulso de *clock*. A configuração do microcontrolador começa, portanto, pelo ajuste do próprio sinal de *clock*, alterando as opções do microcontrolador para uma maior precisão. O microcontrolador conta com um *clock* próprio, mas, conectando um cristal ao dispositivo e fazendo as devidas alterações no *software*, valores mais precisos são obtidos. A escolha de qual a fonte de *clock* é feita no arquivo system_LPC11xx.c, em #define MAINCLKSEL_Val, que altera o registrador de fonte de *clock*.

A escolha para o projeto foi 0x03 - saída do PLL - pois, com o cristal e o PLL, é possível que se tenha uma frequência precisa, maior que a frequência própria do ARM. O PLL recebe, como entrada, a frequência do cristal, que vale 12 MHz e, através do registrador de controle do PLL, tem sua saída multiplicada, conforme a equação (5.1) [6].

Tabela 5.1: Registrador de *clock* principal - MAINCLKSEL

Valor	Seleção
0x00	Oscilador IRC
0x01	Entrada do PLL
0x02	<i>Watch Dog</i>
0x03	Saída do PLL

$$\text{SYS_PLLCLKOUT} = f_{XTAL} \times (\text{SYSPLLCTRL_Val} \& 0x1F) + 1) \quad (5.1)$$

Onde SYSPLLCTRL_Val é outro registrador de baixo nível, cujo valor é formado pelos *bits* de M e P, obtidos através do seguinte algoritmo:

1 - Definir a frequência $f_{clock\ in}$: Dado que

$$f_{clock} = M \times f_{clock_{in}} = \frac{FCCO}{2P} \quad (5.2)$$

$f_{clock\ in}$ é a frequência de *clock* que alimenta o PLL, ou seja, a frequência do cristal, 12 MHz.

2 - Calcular o M inteiro mais próximo do resultado da divisão de f_{clock} (frequência desejada) por $f_{clock\ in}$:

$$M = \frac{50MHz}{12MHz} \approx 4$$

3 - Calcular a frequência corrigida

$$f'_{clock} = M \times f_{clock\ in} = 4 \times 12MHz = 48\text{ MHz}$$

4 - Encontrar FCCO tal que $FCCO = 2 \times P \times f_{clock}$, onde P pode assumir os valores 1, 2, 4 ou 8, FCCO vale entre 156 e 320 MHz [6]:

Tabela 5.2: Cálculo do fator P, para o PLL

P	FCCO (MHz)
1	96
2	192
4	384
8	768

Pela tabela 5.2, tem-se que o valor de P referente ao projeto é P=2, por ser FCCO = 192 MHz o único menor que 320 e maior que 156 MHz. Deve ser atribuído ao registrador SYS_PLLCLKOUT o seguinte: seus 4 *bits* menos significativos equivalem ao valor de M-1 ($4-1 = 3 = 0011_2$) e seus quinto e sexto *bits* são o valor de P ($2=10_2$), formando o seguinte valor: $010\ 0011_2 = 0x23$. Como $0x23 \& 0x1F = 0x03$, usando a equação (5.1), a frequência obtida é de 48 MHz.

5.2 Contagem do tempo

O microcontrolador tem registradores que operam em diferentes escalas de contagem de *clock*. Primeiramente, tem-se o *Prescale Counter* ou PC, que é incrementado a cada pulso de *clock*, variando de 0 a um valor máximo de pulsos contados. Este valor máximo é determinado por um outro registrador, o *Prescale Register*, PR, que possui 32 *bits* [6], ou seja, possui valor máximo de 4.294.967.295. Quando o valor armazenado no PC exceder o valor de PR, PC retorna a zero e o registrador *Time Counter* (TC) é incrementado, como ilustra a figura 5.1. O tempo medido é dado pela equação (5.3).

$$t = PR \times \frac{1}{f_{clock}} \times TC + PC \times \frac{1}{f_{clock}} \quad (5.3)$$

Há, ainda, o *Time Counter Register* (TCR) que, possui 32 *bits*, mas só os *bits* 0 e 1 são acessáveis pelo usuário. O *bit* 0, quando igual a 1, permite a contagem em TC e PC, importante para “pausar” a contagem de tempo, se desejado. O *bit* 1, se

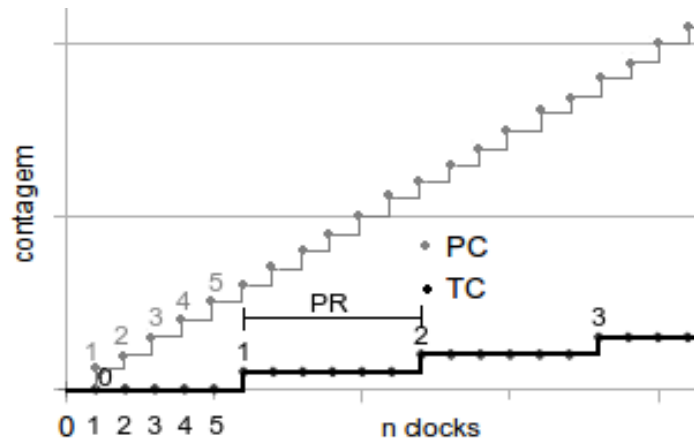


Figura 5.1: Comportamento dos diferentes registradores

for deixado em 1, mantém os valores de TC e PC em zero e sua utilidade é zerar a contagem de tempo.

5.3 Comparação de tempo

Para verificar se o tempo decorrido, medido conforme a seção 5.2, é maior que o intervalo de 3,5 caracteres de silêncio, são utilizados os registradores *Match Register*(MR) e *Match Control Register* (MCR) da seguinte maneira: se o valor obtido for maior que o escrito em MR, a ação correspondente ao valor carregado no MCR será realizada. Esta ação pode ser a geração de uma interrupção no microcontrolador ou a alteração dos registradores TC ou TCR, que, para fins práticos, para ou zera a contagem.

5.4 Resumo dos registradores

5.4.1 Prescale Counter - PC

Este registrador é incrementado a cada pulso de *clock*.

5.4.2 Prescale Register - PR

Define o valor máximo do PC. Quando PC alcança este valor, vai a 0.

5.4.3 *Time Counter* - TC

Contador que é incrementado a cada vez que PC alcançar PR.

5.4.4 *Time Counter Register* - TCR

Tabela 5.3: Registrador contagem de tempo - TCR

<i>Bit</i>	<i>Ação</i>
0	Habilita (quando igual a 1) ou desabilita (0) a contagem em TC e PC
1	Zera PC e TC até que seu valor retorne a zero

Fazendo uma analogia, TCR[0] atua como uma chave e TCR[1], como um botão. TCR[0] liga ou pausa o cronômetro e TCR[1] zera.

5.4.5 *Match Register* - MR

Valor pré-determinado a ser comparado com o TC. Quando TC for igual ao valor carregado em MR, a ação correspondente ao que estiver carregado no *Match Controller Register* será executada.

5.4.6 *Match Control Register* - MCR

O valor colocado neste registrador define a ação quando o *Match Register* atinge o valor do *Timer Counter*.

Tabela 5.4: *Match Controller Register* - MCR

<i>Bit</i>	<i>Ação</i>
0	Gera uma interrupção (altera outro pino do ARM)
1	Zera TC
2	Para a contagem, forçando o valor do bit menos significativo do TCR em 0.

5.5 Programação

Apresentados os principais registradores de manipulação de tempo, podem ser feitas operações como iniciar uma contagem de tempo, parar esta contagem, zerá-la e comparar com o tempo de silêncio característico do RTU.

```
void iniciaCronometro(){
    LPC_TMR32B0->MCR = 0x04;
    LPC_TMR32B0->TCR = 0x01;
}

float zeraCronometro(float tempo){
    LPC_SYSCON->SYSAHBCLKCTRL |= (1<<9);
    LPC_TMR32B0->TCR = 0x11;
    LPC_TMR32B0->PR  = 1000;
    LPC_TMR32B0->IR   = 0xff;
    tempo=0.0;
    return tempo;
}

float pausaCronometro(float tempo){
    LPC_TMR32B0->TCR = 0x00;
    LPC_TMR32B0->PR  = 1000;
    tempo=((float )((LPC_TMR32B0->PC)+(LPC_TMR32B0->PR)*
                    (LPC_TMR32B0->TC)) / SystemCoreClock );
    tempo=tempo*1000.0;
    return tempo;
}
```

Também foram desenvolvidas funções para espera, que funcionam prendendo o programa até que um dos registradores de tempo mude de estado. Essas funções são usadas, tanto na programação da resposta RTU (que deve conter os mesmos marcadores de silêncio) do dispositivo, como também na programação do *display*.

```
int esperaus(double useconds){
```

```

    uint32_t us = (int) useconds +1;
    LPC_SYSCON->SYSAHBCLKCTRL |= (1<<9);
    LPC_TMR32B0->TCR = 0x03;
    LPC_TMR32B0->PR = 0x00;
    LPC_TMR32B0->MR0 = us * ((SystemCoreClock/
                               (LPC_TMR32B0->PR+1))/ 1000000);
    LPC_TMR32B0->MCR = 0x04;
    LPC_TMR32B0->TCR = 0x01;
    while (LPC_TMR32B0->TCR & 0x01) ;
    return 0;
}

int esperams(int mseconds){
    uint32_t us = mseconds;
    LPC_SYSCON->SYSAHBCLKCTRL |= (1<<9);
    LPC_TMR32B0->TCR = 0x03;
    LPC_TMR32B0->PR = 0x00;
    LPC_TMR32B0->MR0 = mseconds * ((SystemCoreClock/
                                     (LPC_TMR32B0->PR+1))/1000);
    LPC_TMR32B0->MCR = 0x04;
    LPC_TMR32B0->TCR = 0x01;
    while (LPC_TMR32B0->TCR & 0x01) ;
    return 0;
}

```

As funções são, respectivamente, para uma entrada em microssegundos e em milissegundos. A figura 5.2 mostra um fluxograma resumido da tomada de decisões do sistema durante a formação do pacote Modbus RTU, com relação à passagem do tempo. Não está especificado na figura a separação do CRC do vetor dados[], indicando apenas o reconhecimento do início e fim da mensagem, ou seja, os silêncios.

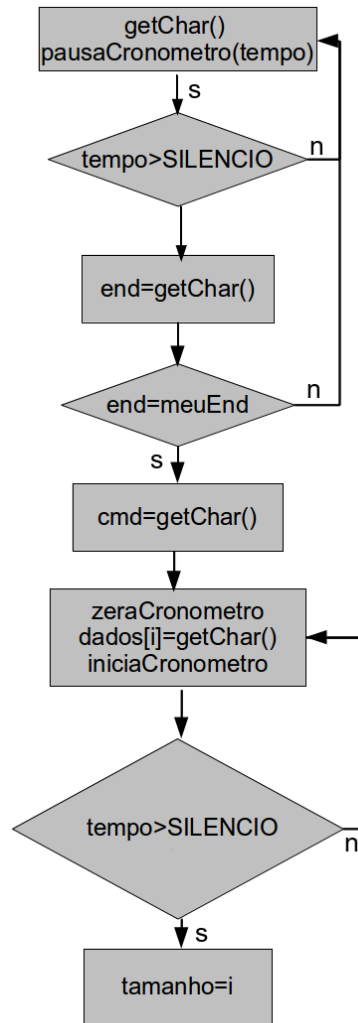


Figura 5.2: Fluxograma resumido do controle de início e fim de mensagem

5.6 Resultados

Os valores dos silêncios, segundo a equação (4.1), foram medidos com um osciloscópio e comparados com os valores desejados (tabela 5.5); todos os tempos obtidos foram superiores ao desejado, o que não há problema, uma vez que o tempo do silêncio deve ser maior ou igual ao valor de 3,5 caracteres, havendo tolerância para valores levemente maiores. A implementação da comunicação RTU pelo método utilizado foi bem sucedida e a *baudrate* escolhida para o projeto foi a de 57600 bits/s.

Tabela 5.5: Comparação entre os tempos de silêncio esperado e obtido

Baudrate (bits/s)	$t_{previsto}$ (μs)	$t_{observado}$ (μs)	erro (μs)	erro relativo
9600	4010,417	4040	29	0,723%
38400	1002,604	1050	47	4,69 %
57600	668,403	712	43	6,43%
115200	334,201	376	41	12,2%

Capítulo 6

Funções Modbus

Montado o pacote de dados descrito no Capítulo 4, deve ser executada a ação correspondente a cada uma das funções Modbus, ou seja, leitura/escrita de valores discretos ou analógicos. Para executar essas funções, o programa deve ler o pacote de dados, reconhecer a função e executar, com o auxílio da pinagem característica da LPC, a ação pedida pelo mestre. Se a ação for bem sucedida, o escravo deve enviar uma mensagem também característica do protocolo Modbus [13]; senão, o escravo deve detectar o erro (que pode ser, por exemplo, função inválida ou erro na verificação da mensagem no LRC/CRC) e avisar ao mestre o problema ocorrido.

O microcontrolador conta com terminais GPIO (*General Purpose In/Out*), para valores digitais, que podem ser configurados como saída (alterar alguma variável externa) ou entrada (receber um valor externo). Esta funcionalidade é usada na leitura e escrita de LEDs, que podem ser acesos ou apagados pelo comando Modbus e também terem seu estado aceso ou apagado determinado remotamente. Também é usado na leitura de botões, cujo valor não é determinado pelo microcontrolador, mas por uma ação externa (pressionado ou solto).

6.1 Formato Geral

6.1.1 Função 0x02 - *Read Input Status*

Esta função tem como objetivo verificar o estado ligado ou desligado de uma entrada discreta do escravo. Para o protótipo, a verificação de um estado liga-

do/desligado pode ser feita com botões presentes no microcontrolador, seguindo a seguinte convenção arbitrária: *bit* 1 para botão apertado e 0, caso contrário. Os botões presentes chaveiam entre 0 e 5 Volts, fornecendo a terminais de leitura de valores digitais do microcontrolador um estado ligado ou desligado.

O formato geral do campo de dados desta função é apresentado na tabela 6.1 [8].

Tabela 6.1: Campo de dados da função 0x01 na solicitação

Endereço do primeiro pino (4 caracteres)	Número total de pinos (4 caracteres)
---	---

O endereço das entradas indicam suas localizações referenciadas ao dispositivo escravo, e podem ser chamados de endereços internos. Estes endereços internos foram definidos do seguinte modo: 0x00 para o botão da esquerda, 0x01 para o superior, 0x02 para o inferior e 0x03 para o da direita, como ilustrado na figura 6.1.

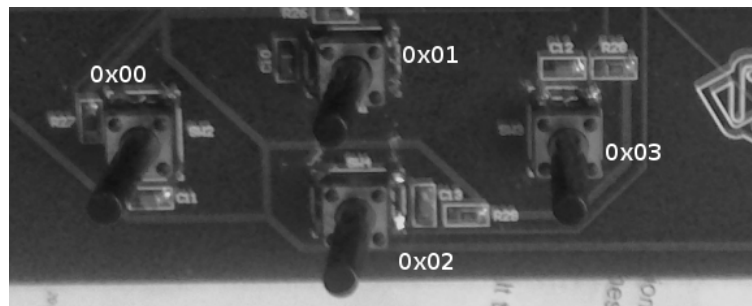


Figura 6.1: Botões do protótipo e os endereços definidos

O número total de entradas solicitadas é sequencial, de maneira que, definido o endereço interno da primeira saída discreta a ser verificada, são analisadas, além dela, todas as saídas de endereço superior até somarem o total especificado pelos 4 caracteres posteriores. A resposta emitida pelo escravo deve conter, em seu campo de dados, as seguintes informações apresentadas na tabela 6.2.

Tabela 6.2: Campo de dados da função 0x01 na resposta

Número de <i>bytes</i> (2 caracteres)	Conteúdo (em hexa) do <i>byte</i> 1 (2 caracteres)	...	Conteúdo do <i>byte</i> N (2 caracteres)
--	---	-----	---

A quantidade de *bytes* necessários é o número de saídas dividido por 8, já que cada saída é representada por um *bit* e 8 *bits* formam 1 *byte*; este número é arredondado para cima, sendo os *bits* adicionais completados com 0. O valor dos *bytes* são calculados da seguinte forma:

$$byte_1 = \sum_{i=0}^8 2^i \times (bit_{end.inicial+i})$$

$$byte_j = \sum_{i=0}^{num.max.} 2^i \times (bit_{end.inicial+8 \times j+i})$$

O resultado enviado é o valor em hexa dos *bytes*. Por exemplo, supondo que serão lidos os valores binários do endereço 19 até o 31 do I/O do microcontrolador, e que sua configuração é tal como a apresentada na tabela 6.3.

Tabela 6.3: Exemplo de estado dos terminais de I/O

Endereço Interno	<i>Bit</i>	Endereço Interno	<i>Bit</i>
19 (0x13)	1	26 (0x1A)	1
20 (0x14)	0	27 (0x1B)	1
21 (0x15)	1	28 (0x1C)	1
22 (0x16)	1	29 (0x1D)	0
23 (0x17)	0	30 (0x1E)	1
24 (0x18)	0	31 (0x1F)	1
25 (0x19)	1		

$$byte_1 = 11001101_2 = \text{CD}_{16}$$

$$byte_2 = 00011011_2 = \text{1B}_{16}$$

Lembrando que, para *bits* representando endereços maiores que 31 (ou 0x1F), são completados zeros. O campo de dados do pacote desta requisição seria “0013 000D”, 0x13 o endereço inicial, 0x0D é o endereço final menos o inicial (incluso), $(31-19)+1 = 13 = 0x0D$. A resposta seria “02CD1B”; 0x02 para dois *bytes* com as informações (CD e 1B) descritas acima. Os exemplos da tabela 6.4 ilustram mensagens completas, fazendo uso da função 0x01, no projeto.

Tabela 6.4: Exemplos do uso da função 0x01

Atividade	Mensagem Ascii	Mensagem RTU
Ler botão esquerdo	:110100000001ED\$0D\$0A	110100000001FF5A
Resposta	:11010101EC\$0D\$0A	110101019488
Ler botões esquerdo e para cima	:110100000002EC\$0D\$0A	110100000002BF5B
Resposta	:11010103EA\$0D\$0A	110101031549
Ler botões para cima, para baixo e direito	:110100010003EA\$0D\$0A	1101000100032F5B
Resposta	:11010107E6\$0D\$0A	11010107148A

6.1.2 Função 0x01 - *Read Coil Status*

Esta função destina-se à leitura de saídas do microcontrolador. A implementação é igual à da função 0x01, porém, são utilizados os *LEDs*; a convenção utilizada é 1 para *LED* apagado e 0, se ele estiver aceso. É possível perceber melhor seu funcionamento na tabela 6.5 [12].

6.1.3 Função 0x04 - *Read Input Registers*

Esta função faz a leitura de dados analógicos, obtidos através do conversor A/D presente conectado ao ARM. O formato da entrada é semelhante ao das funções 0x01 e 0x02: 4 caracteres para o endereço do primeiro registrador analógico e 4 para a quantidade. A resposta implementada usou 2 *bytes* (16 *bits*) para cada registrador, ficando na forma mostrada na tabela 6.6. Um exemplo do uso desta

Atividade	Mensagem Ascii	Mensagem RTU
Ler LED vermelho	:110200340001B8\$0D\$0A	110200340001FA94
Resposta	:11020100EC\$0D\$0A	11020100A548
Ler LEDs verde e amarelo	:11020102EA\$0D\$0A	110200350002EB55
Resposta	:110200340001B8\$0D\$0A	110201016488
Ler LEDs vermelho, verde e amarelo	:11020104E8\$0D\$0A	1102003400037B55
Resposta	:110200340003B6\$0D\$0A	11020103E549

Tabela 6.5: Exemplos do uso da função 0x02

função é apresentado na tabela 6.7. Nele, são pedidos os valores dos registradores 0x02 e 0x03, e verifica-se que estão com os valores 0x00FF e 0x0145.

Tabela 6.6: Campo de dados da função 0x03 na resposta

Número de <i>bytes</i> (2 caracteres)	Conteúdo do registrador 1 (4 caracteres)	...	Conteúdo do registrador N (4 caracteres)
--	---	-----	---

Tabela 6.7: Exemplo do uso da função 0x03

Atividade	Mensagem Ascii	Mensagem RTU
Ler registradores 02 e 03	:110300020002E8\$0D\$0A	110300020002675B
Resposta	:11030400FF0145A3\$0D\$0A	11030400FF01451BA1

6.1.4 Função 0x03 - *Read Holding Registers*

Idêntica à função 0x04.

6.1.5 Função 0x05 - *Force Single Coil*

É a função de mais simples implementação, responsável por definir um valor para uma saída (no protótipo, um *LED*). O formato do seu vetor de dados está representado na tabela 6.8, onde “Valor” é 0xFF para *bit* 1 e 0x00 para *bit* 0. Como mencionado anteriormente, adotou-se *bit* 0 para *LED* aceso, ou seja, 0xFF. A resposta é idêntica à pergunta, se não houver erros. Vale mencionar que o endereço interno dos *LEDs* foram escolhidos com base na pinagem de I/O do ARM. A alteração de variáveis de *hardware* é feita com o auxílio de macros definidas pelo programa LPCXpresso.

Tabela 6.8: Campo de dados da função 0x05 na pergunta

Endereço Interno do Pino (4 caracteres)	Valor (2 caracteres)	00 (2 caracteres)
--	-------------------------	----------------------

Consultando o manual [6], pode-se controlar o estado dos *LEDs* com a seguinte função:

```
void GPIOSetValue( uint32_t portNum ,
                    uint32_t bitPosi ,
                    uint32_t bitVal )
{
    switch (portNum){
        case 0:
            LPC_GPIO0->MASKED_ACCESS[(1<<bitPosi)] = (bitVal<<bitPosi);
            break;
        case 1:
            LPC_GPIO1->MASKED_ACCESS[(1<<bitPosi)] = (bitVal<<bitPosi);
            break;
        case 2:
            LPC_GPIO2->MASKED_ACCESS[(1<<bitPosi)] = (bitVal<<bitPosi);
            break;
        case 3:
```

```

        LPC_GPIO3->MASKED_ACCESS[(1<<bitPosi)] = (bitVal<<bitPosi);
        break;
default:
        break;
}
}

```

Explicitamente, para os *LEDs*:

```

char GPIOSetMapValue(Uint16 endCoil, uint32_t onoff){
    switch (endCoil){
        case 0x34://led vermelho
            GPIOSetValue(1,1,!onoff);
            return 0;
        case 0x35://led verde
            GPIOSetValue(1,2,!onoff);
            return 0;
        case 0x36://led amarelo
            GPIOSetValue(3,0,!onoff);
            return 0;
        default:
            return 2;//erro: pino nao encontrado
    }
}

```

Na tabela 6.9, apresentam-se exemplos de uso da função 0x05.

Tabela 6.9: Exemplos do uso da função 0x05

Atividade	Mensagem Ascii	Mensagem RTU
Acender <i>LED</i> vermelho	:110500340000B6\$0D\$0A	1105003400008E94
Resposta	:110500340000B6\$0D\$0A	1105003400008E94
Apagar <i>LED</i> vermelho	:11050034FF00B7\$0D\$0A	11050034FF00CF64
Resposta	:11050034FF00B7\$0D\$0A	11050034FF00CF64

6.1.6 Função 0x06 - *Preset Single Register*

Responsável por atribuir um valor a um registrador analógico de forma isolada. Para o protótipo, foi utilizado um *display* de forma a obter o retorno do valor escrito. A função desenvolvida obtém um valor do mestre e o escreve usando 4 trechos do *display*, sendo informada, também, a posição no *display* onde será posicionado. A figura 6.2 mostra as posições do *display*:

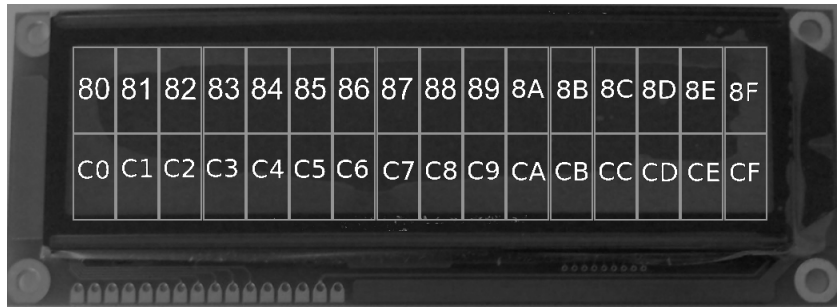


Figura 6.2: Endereços das posições de caracteres do *display*

Seu campo de dados deve possuir o endereço onde haverá o *byte* mais significativo da sequência, e o valor a ser escrito, conforme tabela 6.10.

Tabela 6.10: Campo de dados do pedido na função 0x06

Endereço Interno no <i>display</i> (4 caracteres)	Valor (4 caracteres)
--	-------------------------

6.1.7 Função 0x0F - *Force Multiple Coils*

Função cuja finalidade é atribuir valores ligado/desligado a uma ou mais saídas discretas, que, como foi dito antes, são *LEDs*, no protótipo desenvolvido. O mestre deve enviar o endereço interno inicial da saída discreta, a quantidade de saídas a serem lidas, de forma sequencial, o número de *bytes* a serem disponibilizados e os valores, seguindo o mesmo cálculo mostrado na seção 6.1.1. O formato de dados da solicitação e da resposta são mostrados nas tabelas 6.12 e 6.13, respectivamente e alguns exemplos são mostrados na tabela 6.14.

Tabela 6.11: Exemplos do uso da função 0x06

Atividade	Mensagem Ascii	Mensagem RTU
Escreve 62513 a partir do trecho 0xC0	:110600C0FA31FE\$0D\$0A	110600C0FA310812
Resposta	:110600C0FA31FE\$0D\$0A	110600C0FA310812
Escreve 64225 a partir do trecho 0x80	:11060080FAE18E\$0D\$0A	11060080FAE1085A
Resposta	:11060080FAE18E\$0D\$0A	11060080FAE1085A

Tabela 6.12: Campo de dados da função 0x0F na solicitação

Primeiro LED (4 caracteres)	Número de <i>LEDs</i> (4 caracteres)	Número de <i>bytes</i> (2 caracteres)	Conteúdo do <i>byte</i> 1 (4 caracteres)	...	Conteúdo do <i>byte</i> N (4 caracteres)
-----------------------------------	--	---	--	-----	--

6.1.8 Função 0x10 - *Preset Multiple Registers*

Esta função permite atribuir valores a um ou mais registradores e foi implementada utilizando o *display*: cada registrador, apesar do tamanho de 2 *bytes*, só pode assumir valores de 0 a 15 (0x0F) e cada registrador é escrito em um dos trechos do *display*. Ou seja, o endereço do registrador foi assumido como sendo o trecho onde se quer escrever o valor, o qual só pode ter 4 *bits*, por questão de implementação. O formato da mensagem emitida pelo mestre é exibido na tabela 6.15 e a resposta exhibe o endereço do primeiro registrador e o número de registradores alterados (tabela 6.16). Um exemplo pode ser visto na tabela 6.17.

Tabela 6.13: Campo de dados da função 0x0F na resposta

Primeiro LED (4 caracteres)	Número de <i>bytes</i> (4 caracteres)
--------------------------------	--

Tabela 6.14: Exemplos do uso da função 0x0F

Mensagem Ascii	
Acende <i>LEDs</i> vermelho, amarelo e verde	:110F00340003030000A6\$0D\$0A
Resposta	:110F00340003A9\$0D\$0A
Apaga <i>LEDs</i> vermelho, amarelo e verde	:110F003400030300079F\$0D\$0A
Resposta	:110F00340003A9\$0D\$0
Mensagem RTU	
Acende <i>LEDs</i> vermelho, amarelo e verde	110F003400030300007ED0
Resposta	110F003400035694
Apaga <i>LEDs</i> vermelho, amarelo e verde	110F003400030300073F12
Resposta	110F003400035694

6.1.9 Erro

Quando uma solicitação não pode ser atendida pelo escravo, ele deve enviar uma mensagem alertando sobre qual erro aconteceu, cabendo ao mestre a decisão de como proceder (mostrar um aviso de erro e reenviar, por exemplo). Esta mensagem é tal que o campo de função é modificado através de seu *bit* mais significativo, que se torna 1. Por exemplo, se ocorrer um erro na função 0x03, o campo de função da resposta será 0x83 (0000 0011 é alterado para 1000 0011). Se a função for 0x10, seu equivalente para erro é 0x90. Em seguida, é enviado o código do erro ocorrido, que será um dentre os mencionados na tabela 6.18. [14]

Por fim, é feito o cálculo da verificação dos dados a serem enviados, formando o pacote da resposta ao erro, que está ilustrado pela tabela 6.19. Um exemplo pode ser visto na tabela 6.20.

Tabela 6.15: Campo de dados da função 0x10 na solicitação

Endereço do 1º Reg. (4 caracteres)	Número de registradores (4 caracteres)	Número de <i>bytes</i> (2 caracteres)	Conteúdo do reg.1 (4 caracteres)	...	Conteúdo do reg.N (4 caracteres)
--	--	---	--	-----	--

Tabela 6.16: Campo de dados da função 0x10 na resposta

Endereço do 1º registrador (4 caracteres)	Quantidade de Registradores (4 caracteres)
--	---

6.2 Implementação

6.2.1 Percorrer o número de *bytes*

Foi explicado na seção 6.1.1 como calcular o número de *bytes* necessários para transmitir informações no protocolo Modbus. Para percorrer esses *bytes*, *bit* por *bit*, foi utilizado o seguinte código:

```
while(indice!=2*N){
    while(quantity&&(multiplier!=9)&&(error!=2)){
        error=GPIOGetMapValue(coilAddress);
        currValue=currValue+error *
            potencia(multiplier);
        getValues[indice]=currValue;
        multiplier++;
        coilAddress++;
        quantity--;
    }
    pacote.dados[indice+2]=
        hex2char(getValues[indice]/0x10);
    pacote.dados[indice+3]=
```

Tabela 6.17: Exemplos do uso da função 0x10

Atividade	Escreve 0xF no trecho 0x80, 0xA, 0xE e 0x1 nos seguintes
Mensagem Ascii	:11100080000408000F000A000E00012B\$0D\$0A
Resposta Ascii	:1110008000045B\$0D\$0A
Mensagem RTU	11100080000408000F000A000E0001630A
Resposta RTU	111000800004C2B2

Tabela 6.18: Fontes de erro Modbus

Código	Erro	Descrição
0x01	Função Inválida	Não existe a função solicitada
0x02	Endereço Inválido	Pedido para um pino ou registrador não encontrado
0x03	Dado Inválido	Erro na verificação dos dados (CRC e LRC)

```

        hex2char ( getValues [ indice ] % 0x10 );
    multiplier = 0;
    indice += 2;
}

```

A variável *error* recebe 0 ou 1 para o valor obtido, ou 2 em caso de erro. O valor do *byte* atual é colocado na variável *currValue*, que tem seu valor alterado a cada *bit* lido. O laço de repetição interno verifica, a cada *bit* lido, se houve erro, se a quantidade total de *bytes* não se encerrou ou se todos os *bits* do *byte* atual foram lidos. O laço externo controla a variação dos *bytes*. ‘N’ é uma variável do tipo *char* que recebe o cálculo do número de *bytes* necessários.

```

N = quantity / 8;
if ( quantity % 8 ) N++;

```

Tabela 6.19: Mensagem Modbus e número de caracteres por bloco

Endereço do dispositivo	Código de Função alterado	Erro	Verificação
(2 caracteres)	(2 caracteres)	(2 caracteres)	(2 <i>bytes</i>)

Tabela 6.20: Exemplos de respostas a erros

Erro	Resposta Ascii	Resposta RTU
Função 0x1F inexistente	:119F014F\$0D\$0A	119F0189F5
<i>LEDs</i> 0x38 e 0x39 inválidos (função 0x0F)	:118F025E\$0D\$0A	118F02C434
Erro de LRC/CRC (função 0x02)	:1182036A\$0D\$0A	118F030164

E, *getValues* é um vetor **uint32_t**, ou seja, inteiros sem sinal de 32 *bits*, de tamanho N, para guardar os *bytes*. Para a comunicação RTU, o código fica levemente modificado:

```

while (indice!=N){
    while (quantity&&(multiplier!=9)&&(error!=2)){
        error=GPIOGetMapValue (coilAddress);
        currValue=currValue+error * potencia (multiplier);
        getValues [ indice]=currValue;
        multiplier++;
        coilAddress++;
        quantity--;
    }
    pacote.dados [ indice+1]=getValues [ indice ];
    multiplier=0;
    indice++;
}

```

6.2.2 Ler I/O

Para leitura de entrada e saída, são usados endereços dos registradores do ARM, que podem ser estudados em seu manual [6]. Os endereços dos itens de entrada e saída são definidos no microcontrolador como pares $(portNum, bitPosi)$ e os usados no protótipo estão na tabela 6.21 [6]. Como visto na seção 6.1.5, com esse mapeamento e utilizando macros definidas pelo LPCXpresso, é possível definir valores para as saídas digitais (funções 0x05 e 0x0F), além de ler seus valores e os de entrada (funções 0x01 e 0x02).

Tabela 6.21: Mapa I/O

Dispositivo	Par $(portNum, bitPosi)$
Botão Esquerdo	(0,7)
Botão Cima	(2,9)
Botão Baixo	(0,8)
Botão Direito	(2,10)
<i>LED</i> Vermelho	(1,1)
<i>LED</i> Verde	(1,2)
<i>LED</i> Amarelo	(3,0)

6.2.3 Escrever no *display*

Consultando exemplos do LPCXpresso para visualização em *display* [5], foi possível basear-se neles para construir as funções de acesso ao *display*. As funções utilizadas foram a de apagar todo o conteúdo escrito - `LimpaLCD()` - escrever um caracter - `writeChar` - e escrever um conteúdo em hexadecimal - `writeHex`. Todas essas funções acessam a função `AtualizaDisplay16x2`, que prepara os registradores do microcontrolador para modificar o conteúdo exibido.

```
void AtualizaDisplay16x2(char rs ,char meuDado){  
    Display16x2 Display ;  
  
    Display.Dado = meuDado ;
```

```

Display.bits.Rs = rs;
GPIOSetValue(0,3,0); //R/W = 0
GPIOSetValue(2,8,0); //EN = 0

GPIOSetValue(0,4,Display.bits.Rs);
GPIOSetValue(2,0,Display.bits.D0);
GPIOSetValue(2,1,Display.bits.D1);
GPIOSetValue(2,2,Display.bits.D2);
GPIOSetValue(2,3,Display.bits.D3);
GPIOSetValue(2,4,Display.bits.D4);
GPIOSetValue(2,5,Display.bits.D5);
GPIOSetValue(2,6,Display.bits.D6);
GPIOSetValue(2,7,Display.bits.D7);
GPIOSetValue(2,8,1); // Enable = 1
esperaus(20.0);
GPIOSetValue(2,8,0); // Enable = 0
}

```

No código acima, *rs* é um *flag* para indicar se a operação realizada é para informar o endereço ou o valor; *meuDado* é o valor a ser escrito; e *Display16x2* é uma estrutura de dados que contém os *bits* de controle do *display* e os dados atuais.

```

Uint32 Dado          : 8; // 0-7
Uint32 Controle      : 4; // 8-11;
struct { // bits    description
    Uint32 D0 : 1; // 0
    Uint32 D1 : 1; // 1
    Uint32 D2 : 1; // 2
    Uint32 D3 : 1; // 3
    Uint32 D4 : 1; // 4
    Uint32 D5 : 1; // 5
    Uint32 D6 : 1; // 6
    Uint32 D7 : 1; // 7
}

```

```

        Uint32 Rw : 1; // 8
        Uint32 Rs : 1; // 9
        Uint32 Enable : 1; // 10
        Uint32 Power : 1; // 11
    } bits;
} Display16x2;

```

A função `LimpaLCD()` é apresentada abaixo:

```

void LimpaLCD(void) {
    AtualizaDisplay16x2(0,0x01);
    esperams(2);
}

```

A função `writeChar` envia o comando para alocar o endereço do *display* onde haverá a escrita e, em seguida, o valor em si, caracter por caracter. Já a função `writeHex` recebe um valor inteiro *value*, de 0 a 15 (base 16), e simplesmente transforma o valor no seu valor correspondente na tabela Ascii.

```

void writeChar(char endereco,char ascii){
    if(endereco != 0) {
        AtualizaDisplay16x2(0,(0x80|endereco));
    }
    AtualizaDisplay16x2(1,ascii);
}

```

```

void writeHex(char address,char value){
    int valueHex=0;
    if(value<10) valueHex=value+0x30;
    else valueHex=87+value;
    writeChar(address,valueHex);
}

```

6.2.4 Cálculo do LRC

Já foi mostrado na seção 4.2 o cálculo e a implementação da LRC. Abaixo, a função para comparar o LRC calculado com o enviado pelo mestre: se forem iguais, executa a função, senão, envia erro 0x03.

```
void getLRC() {
    meuPacote.lrc0=meuPacote.dados[meuPacote.tamanho-2];
    meuPacote.lrc1=meuPacote.dados[meuPacote.tamanho-1];
    meuPacote.tamanho=meuPacote.tamanho-2;
    lrcCalc=calculaLRC(meuPacote);
    if (lrcCalc/0x10==char2hex(meuPacote.lrc0) &&
        lrcCalc%0x10==char2hex(meuPacote.lrc1))
        executa();
    else erroAscii('0','3',meuPacote);
}
```

6.2.5 Cálculo do CRC

Seu cálculo está na seção 4.3 e a implementação da sua chamada consta a seguir.

```
void getCRC() {
    pacote.crc=256*pacote.dados[pacote.tamanho-2]+
                pacote.dados[pacote.tamanho-1];
    pacote.tamanho=pacote.tamanho-2;
    crcRX=calculaCRC(pacote);
    if (crcRX==pacote.crc)
        executaRTU();
    else erro(0x03,pacote);
}
```

6.2.6 Resposta sem erro

Quando não há erros, as respostas seguem os padrões mencionados na seção 6.1 e devem conter o endereço do dispositivo, a função, o campo de dados e a verificação de erros, além dos caracteres iniciador e terminadores, no caso Ascii.


```

SendChar( ' : ' );
SendChar( pacote . end0 );
SendChar( pacote . end1 );
SendChar( pacote . cmd0 );
SendChar( pacote . cmd1 );
pacote . dados [0] = hex2char (N/0x10 );
pacote . dados [1] = hex2char (N%0x10 );
pacote . tamanho = 2*(N+1);
for ( indice = 0; indice < pacote . tamanho; indice++) {
    SendChar( pacote . dados [ indice ] );
}
Uint16 lrc = calculaLRC ( pacote );
pacote . lrc0 = lrc / 0x10;
SendChar( hex2char ( pacote . lrc0 ) );
pacote . lrc1 = lrc % 0x10;
SendChar( hex2char ( pacote . lrc1 ) );
SendChar( 0x0A );
SendChar( 0x0D );

```

Para o protocolo RTU, não há os caracteres de início e fim de mensagem e há menos chamadas à função de envio de caracteres (pelo motivo da economia de *bytes* já explicado na seção 4.3), entretanto é preciso fazer controle do tempo, usando os conceitos, registradores e funções explicados na seção 5.5. A variável *silencio* é calculada seguindo a equação (4.1).

```

esperaus ( silencio );
SendChar( pacote . end );
SendChar( pacote . cmd );
pacote . dados [0] = N;
pacote . tamanho = (N+1);
pacote . crc = calculaCRC ( pacote );
for ( indice = 0; indice < pacote . tamanho; indice++)
    SendChar( pacote . dados [ indice ] );

```

```

SendChar((pacote.crc)/0x100);
SendChar((pacote.crc)%0x100);
esperaus(silencio);

```

6.2.7 Resposta com erro

A indicação de erro só necessita alterar o campo de função. O código do erro é dado como entrada, indicado pela implementação das várias funções do programa.

```

void erroAscii(char erro0 , char erro1 , pacote_ascii pacote){
    SendChar(' : ');
    SendChar(pacote.end0);
    SendChar(pacote.end1);
    if(pacote.cmd0>=0x31) pacote.cmd0='9';
    else pacote.cmd0='8';
    SendChar(pacote.cmd0);
    SendChar(pacote.cmd1);
    pacote.dados[0]=erro0;
    SendChar(pacote.dados[0]);
    pacote.dados[1]=erro1;
    SendChar(pacote.dados[1]);
    pacote.tamanho=2;
    Uint16 lrc=calculaLRC(pacote);
    pacote.lrc0 = lrc/0x10;
    SendChar(hex2char(pacote.lrc0));
    pacote.lrc1 = lrc%0x10;
    SendChar(hex2char(pacote.lrc1));
    SendChar(0x0A);
    SendChar(0x0D);
}

```

Capítulo 7

Realização do experimento

7.1 SCADA

Os chamados Sistemas de Supervisão e Aquisição de Dados (SCADA) permitem o monitoramento de forma amigável e à distância de equipamentos industriais, isto é, exibem informações gráficas, permitem o controle de parâmetros físicos, como, por exemplo, gráficos e botões liga/desliga, guardam relatórios, disparam alarmes, entre outras facilidades. Um diagrama de blocos simplificado do sistema é mostrado na figura 7.1. O sensoreamento é feito pelo microcontrolador que usa a conexão via cabo serial, para receber as ordens do SCADA, processa as informações sobre os pinos de entrada e saída de dados sobre um objeto de medida e retorna ao SCADA a confirmação das alterações e valores lidos.

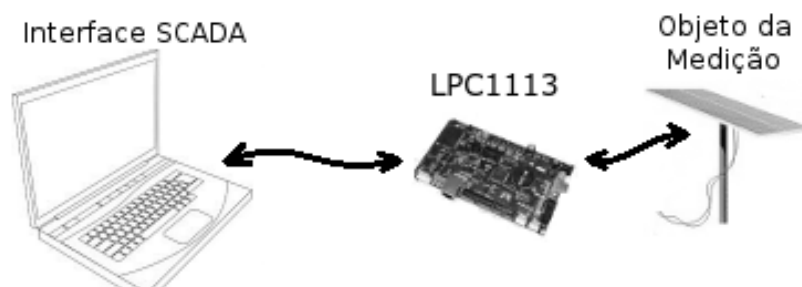


Figura 7.1: Diagrama de blocos do sistema conectado à *internet*

Podem ser citados como opções de SCADA o Elipse SCADA, iaFox SCADA++, pvbrowser e ScadaBR. Consultando-se as páginas dos fabricantes, podem ser en-

contradas informações sobre os protocolos compatíveis e facilidades, como banco de dados, IHM e compatibilidade com sistemas operacionais. O Elipse exige o Windows como sistema operacional [15], o que é uma desvantagem pois se procura um sistema independente de plataforma e, de preferência, que prestigie sistemas de código livre, apesar de sua dominância no mercado nacional [16]. O SCADA++ fornece compatibilidade com banco de dados, porém, é pouco utilizado. As duas melhores opções avaliadas foram o pvbrowser e o ScadaBR, ambos de filosofia de código livre [17],[18], com suporte ao protocolo Modbus e TCP/IP e interface similar a uma navegação, o primeiro usando Qt e o segundo, Java JDK. Então, optou-se pela opção nacional, com suporte mais acessível (fórum em português ou diretamente com os desenvolvedores) e manuais de instalação e uso didáticos.

7.2 Uso do ScadaBR

O programa necessita a instalação do Java 6 (JDK 1.6) e o Apache Tomcat 6, o qual permite o acesso ao programa como se fosse uma página *Web*, através de um navegador, pelo endereço localhost:8080/Scadabr . Além disso, devem ser instaladas as bibliotecas RxTx para o Java e o ScadaBR propriamente dito. A obtenção de dados utiliza os conceitos de *data source* e *data points* .

Ao criar um *data source*, são informados o protocolo, quais conexões físicas e alguns parâmetros, como tempo de espera (*timeout*) e o intervalo de tempo entre uma solicitação e outra, funcionando como um período de amostragem do SCADA. A cada intervalo desse, são atualizados os valores do painel de variáveis chamado *watch list*. Dado um *data source*, podem ser criados *data points* referentes, o que significa, na prática, acesso a informações do microcontrolador, como pinos de I/O, e que estarão disponíveis para serem mostrados na *watch list*. As figuras 7.2 e 7.3 ilustram o uso desses conceitos. Quando há algum erro, seja ele informado pelo escravo (através das funções apresentadas no Capítulo 6), ou expiração do tempo de espera, o ScadaBR mostra em sua *watch list* a mensagem “O valor do ponto pode não ser confiável”.

Para configurar a rede Modbus deste projeto, foram utilizados os seguintes parâmetros: protocolo Modbus Ascii e RTU, *baudrate* de 57,6 kbps, porta serial COM simulada pelo adaptador USB-RS232 (dependendo da porta USB, o número da COM varia). Além disso, deve ser informado o endereço do escravo, definido como 0x11 pelo programa do projeto. Para a medida do A/D, em particular, foi configurada, também, o multiplicador e o *offset*, que indicam, já na interface do ScadaBR, o valor lido pelo microcontrolador multiplicado e somado pelos valores das equações (3.1), que permitem ao microcontrolador ler valores de -12V a 12V, e (3.2), da conversão para o valor digital.

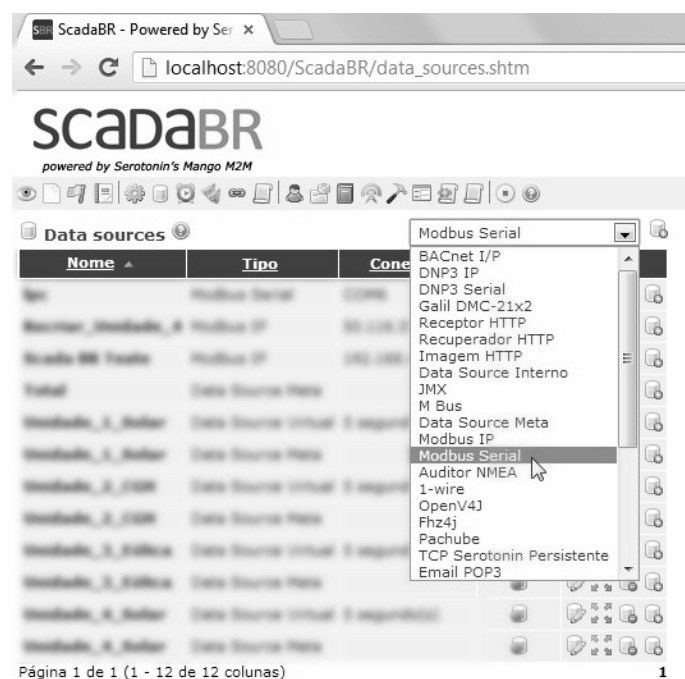


Figura 7.2: Criação de um *data source*

7.3 Resultados

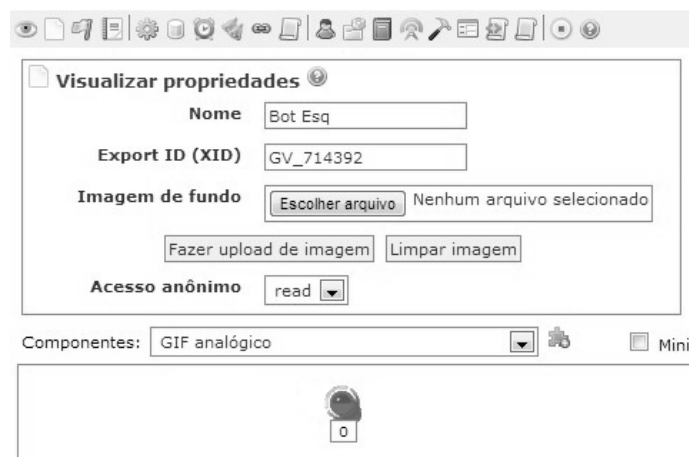
Através de recursos gráficos do ScadaBR, foi possível obter o comportamento de dispositivos de I/O, mostrando, com figuras, se, por exemplo, um botão está ou não acionado. Por exemplo, a figura 7.4 mostra o comportamento de um dos botões do microcontrolador, exibindo uma luz apagada para o botão não pressionado e o valor 0 logo abaixo (usando a função Modbus 0x01). Também é possível apagar ou acender um LED através da interface do ScadaBR (através da função 0x05), indo



Nome	Potenciometro
Export ID (XID)	DP_566744
Id do escravo	17
Faixa do registro	Registrador holding
Tipo de dados modbus	Inteiro de 2 bytes sem sinal
Offset (baseado em 0)	0
Bit	0
Número de registradores	0
Codificação de caracteres	ASCII
Configurável	<input checked="" type="checkbox"/>
Multiplicador	1
Aditivo	0

Figura 7.3: Criação de um *data point*

no painel de variáveis, como pode ser visto pela figura 7.5. Foi encontrada como uma desvantagem do ScadaBR o fato de que só um LED ou botão, ou seja, apenas uma variável digital pode estar associada a um *data point*. A leitura e alteração de variáveis com endereço sequencial usando apenas uma solicitação não é possível pelo painel do ScadaBR, apesar do protocolo Modbus permitir isto através das funções 0x01, 0x02 e 0x0F.



Nome	Bot Esq
Export ID (XID)	GV_714392
Imagem de fundo	Escolher arquivo Nenhum arquivo selecionado
Fazer upload de imagem	Limpar imagem
Acesso anônimo	read

Componentes: GIF analógico Mini

0

Figura 7.4: Acompanhamento em tempo real de uma variável discreta

Assim como para valores digitais, foi feita a leitura e escrita de valores analógicos através da função 0x06. Para a escrita, foi usado o *display*, de maneira que o ScadaBR solicita um valor ao microcontrolador que é mostrado no *display*. O resultado

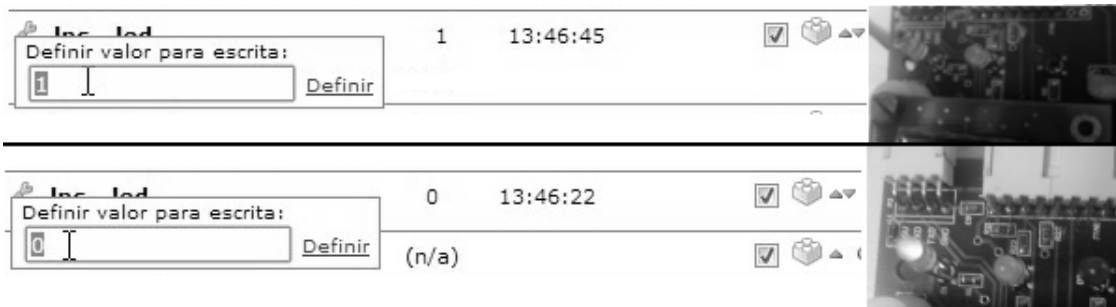


Figura 7.5: Escrita de variáveis discretas e acendimento de LED

desta operação é ilustrado pela figura 7.6. Já a leitura foi feita com o A/D, com o ScadaBR devendo mostrar a tensão de entrada do sistema de conversão.



Figura 7.6: Escrita de valores contínuos

7.3.1 Ajuste do SCADA usando parâmetros do circuito

Com a *watch list* do ScadaBR podem ser obtidos gráficos do valor medido pelo A/D. O ScadaBR, ao trabalhar com valores analógicos, pede como entrada um multiplicador e um aditivo, como mostra a figura 7.3. Por padrão, o multiplicador é igual a 1 e o aditivo, 0, e com esses valores, foi obtida a tabela 7.1. Os valores de entrada são modificados pelo circuito pelas equações (3.1) e (3.2).

Tabela 7.1: Valores digitais obtidos no ScadaBR

v_{in}	v_{out}	valor digital previsto	valor digital obtido	erro relativo (%)
-12V	0	0	0	0
-10,3V	0,06V	18	1	95
-9V	0,22V	67	56	16
-6V	0,58V	179	183	2,3
0V	1,3V	403	440	9,2
6V	2,0V	627	700	12
9V	2,4V	739	830	12
12V	2,7V	851	960	13

Para que o ScadaBR mostre os valores analógicos, pode-se optar por usar diretamente os parâmetros da digitalização (equação (3.2)) e do condicionamento (equação (3.1)) ou obter esses parâmetros pelo método dos mínimos quadrados.

Usando os parâmetros do circuito, os valores do fator aditivo foi igual a -10,9 e o multiplicativo foi de 0,027, conforme o cálculo da equação (7.1). A comparação entre os valores de entrada e os valores exibidos pelo ScadaBR, após usar os fatores, é dada na tabela 7.2.

$$\text{Valor Digital} = \frac{1023}{3,0} \left(1,3 + \frac{v_{in}}{8,3}\right) \approx 403 + 37v_{in}$$

$$v_{in} = \frac{\text{Valor Digital} - 403}{37} = 0,027\text{Valor Digital} - 10,9 \quad (7.1)$$

7.3.2 Ajuste do SCADA calibrando os resultados

Usando o método dos mínimos quadrados, usando os valores obtidos na tabela 7.1 como eixo y e os valores de v_{in} como x, foram encontrados os seguintes coeficientes para a curva $y=mx+y_0$: $m \approx 43,02$ e $y_0 \approx 442,4$. O gráfico mostrando a melhor reta de aproximação dos dados encontra-se na figura 7.7.

Tabela 7.2: Valores medidos usando parâmetros do circuito

\mathbf{V}_{in}	\mathbf{V}_{lido}
-12,0V	-10,9V
-10,3V	-10,9V
-9,0V	-9,4V
-6,0V	-6,0V
0,0V	1,0V
6,0V	8,0V
9,0V	11,5V
12,0V	15,0V

$$v_{\text{medido}} = \frac{1}{m}(\text{Valor Digital} - y_0) = 0,023\text{Valor Digital} - 10,3 \quad (7.2)$$

Para obter a conversão do valor digital para o analógico, usa-se a expressão da equação (7.2) e os resultados estão na tabela 7.3. O acompanhamento gráfico gerado pelo ScadaBR pode ser visto na figura 7.8. O tempo de amostragem do ScadaBR foi de 5 segundos.

Tabela 7.3: Valores medidos usando o método dos mínimos quadrados

\mathbf{V}_{in}	\mathbf{V}_{lido}
-12,0V	-10,3V
-10,3V	-10,3V
-9,0V	-9,0V
-6,0V	-6,0V
0,0V	0,06V
6,0V	6,0V
9,0V	9,0V
12,0V	12,0V

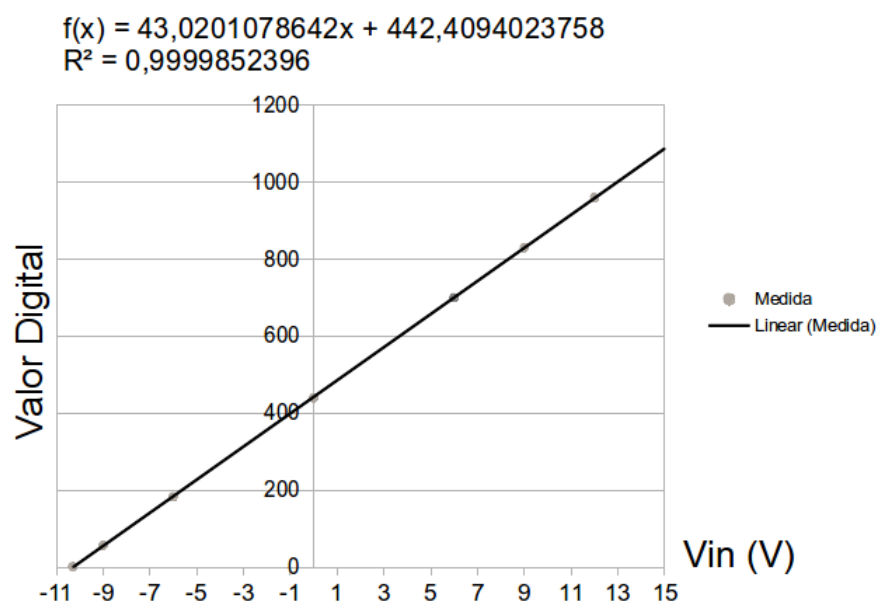


Figura 7.7: Reta de calibração dos valores digitais

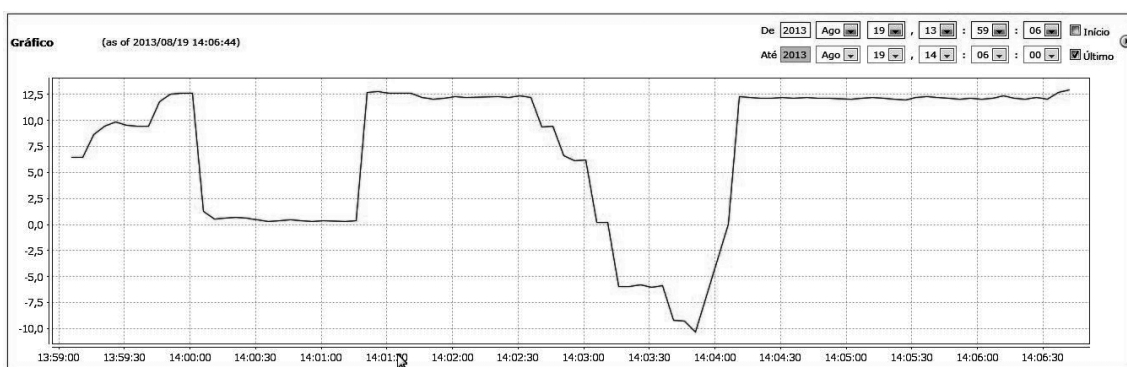


Figura 7.8: Gráfico de valores de tensão (em Volts) mostrados pelo ScadaBR

Capítulo 8

Conclusão

O projeto mostrou que é possível desenvolver um sistema de baixo custo, utilizando *softwares* gratuitos e *kits* de desenvolvimento, obtendo bom retorno visual e fazendo uso de um protocolo de grande uso no mercado de automação que não possui bibliotecas prontas para uso livre. A comunicação foi feita com um cabo serial, e o protocolo foi implementado com sucesso, permitindo que as mensagens enviadas pelo programa SCADA fossem corretamente interpretadas, executadas e respondidas, incluindo-se tratamento de erros. Os valores medidos usando como parâmetros os componentes do circuito tiveram erro da ordem de 20% para os valores positivos e de 5% para os negativos, exceto pela limitação de menor medida possível igual a -10,3V. O erro das medidas pôde ser diminuído usando calibração e método dos mínimos quadrados, com variação nula entre as medidas feitas na entrada e saída do sistema. Além disso, as funcionalidades de entradas e saídas discretas do SCADA mostraram de forma visual que o microcontrolador estava processando corretamente as informações, através das várias funções Modbus descritas no projeto.

A evolução do trabalho pode ser feita com uma rede Modbus com dois ou mais escravos, cada um com seu endereço, usando o meio físico RS-485. Os LEDs e botões podem ser substituídos por elementos que interajam com outros equipamentos, modificando o estado de relés, por exemplo.

Para o prosseguimento do trabalho, deve-se implementar o protocolo Modbus TCP que aproveita grande parte do já implementado Modbus RTU, sendo necessário configurar o microcontrolador para funcionar como um servidor, recebendo pacotes

TCP/IP e encapsulando as mensagens Modbus RTU em um pacote TCP/IP. Outra melhoria, seguindo a tendência do mercado, é a configuração de um SCADA para funcionar como um aplicativo em *smartphones*, sendo necessária também configurar o cliente TCP/IP. Para operações em campo, deve ser levada em conta a possibilidade de problemas com sincronismo e como revertê-los.

Há também a possibilidade de se modificar o circuito de condicionamento do sinal para receber frequências mais altas, alterando o primeiro estágio do circuito da figura 3.2 para um atenuador (divisor resistivo com um *buffer*) para evitar problemas com oscilação, sendo importante, também, verificar o comportamento da fase na resposta em frequência.

Referências Bibliográficas

- [1] “The Register: Sci/Tech News for the World”, http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers, acessada em 18/03/2013.
- [2] “ARM - The architecture For The Digital World”, <http://www.arm.com>, acessada em 28/10/2012.
- [3] “The Modbus Organization”, <http://www.modbus.org>, acessada em 18/03/2013.
- [4] LUGLI, A. B., “Uma visão do protocolo industrial Profinet e suas aplicações”, <http://www.inatel.br>, acessada em 29/08/2013.
- [5] “NXP Semiconductors”, <http://www.nxp.com>, acessada em 19/03/2013.
- [6] “UM10398, LPC111x/LPC11Cxx User manual Rev. 11”, 2012.
- [7] “Modbus description”, <http://www.modbustools.com/modbus.asp>, acessada em 18/03/2013.
- [8] “Simply Modbus - Data Communication Test Software”, <http://www.simplymodbus.ca>, acessada em 19/03/2013.
- [9] “Comandos para Indicadores de Pesagem no Protocolo Modbus-RTU Linha 3000C / 3000C.S., ALFA INSTRUMENTOS ELETRÔNICOS LTDA”, http://www.alfainstrumentos.com.br/manuais/comunicacao/modbus_manual.pdf, acessada em 18/03/2013.
- [10] SALOMON, D., *Data Compression - The Complete Reference*. Springer, 4a Edição.

- [11] D.A.GODSE, A.P.GODSE, *Microprocessors & Microcontroller Systems*. Technical Publications, 1a edição.
- [12] “Modbus Poll User manual”, http://www.modbustools.com/PI_MBUS_300.pdf, acessada em 18/03/2013.
- [13] “MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b”, http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf, acessada em 19/03/2013.
- [14] “Modbus Tutorial: Control Solutions”, http://www.csimn.com/CSI_pages/Modbus101.html, acessada em 19/03/2013.
- [15] “Elipse Knowledgebase”, <http://kb.elipse.com.br/>, acessada em 19/03/2013.
- [16] “1º mini-Workshop, Nivelamento Técnico e Levantamento de Requisitos - Projeto FINEP/SEBRAE ScadaBR - MCA, Unis, Conetec, UFSC - 05 de novembro de 2009”, <http://tinyurl.com/apresentaScadaBr>, acessada em 19/03/2013.
- [17] “pvbrowser - The Process Visualization Broser. HMI and Scada for every platform.”, <http://pvbrowser.de>, acessada em 19/03/2013.
- [18] “ScadaBR - Automação para Todos”, <http://www.scadabr.com.br>, acessada em 19/03/2013.
- [19] “Modbus, Modbus Software, Modbus RTU Modbus ASCII”, <http://www.modbustools.com>, acessada em 19/03/2013.

Apêndice A

Tabela Ascii

000 ₁₀	00 ₁₆	(nul)	023 ₁₀	17 ₁₆	(etb)	046 ₁₀	2E ₁₆	.
001 ₁₀	01 ₁₆	(soh)	024 ₁₀	18 ₁₆	(can)	047 ₁₀	2F ₁₆	/
002 ₁₀	02 ₁₆	(stx)	025 ₁₀	19 ₁₆	(em)	048 ₁₀	30 ₁₆	0
003 ₁₀	03 ₁₆	(etx)	026 ₁₀	1A ₁₆	(eof)	049 ₁₀	31 ₁₆	1
004 ₁₀	04 ₁₆	(eot)	027 ₁₀	1B ₁₆	(esc)	050 ₁₀	32 ₁₆	2
005 ₁₀	05 ₁₆	(enq)	028 ₁₀	1C ₁₆	(fs)	051 ₁₀	33 ₁₆	3
006 ₁₀	06 ₁₆	(ack)	029 ₁₀	1D ₁₆	(gs)	052 ₁₀	34 ₁₆	4
007 ₁₀	07 ₁₆	(bel)	030 ₁₀	1E ₁₆	(rs)	053 ₁₀	35 ₁₆	5
008 ₁₀	08 ₁₆	(bs)	031 ₁₀	1F ₁₆	(us)	054 ₁₀	36 ₁₆	6
009 ₁₀	09 ₁₆	(tab)	032 ₁₀	20 ₁₆	(espaço)	055 ₁₀	37 ₁₆	7
010 ₁₀	0A ₁₆	(lf)	033 ₁₀	21 ₁₆	!	056 ₁₀	38 ₁₆	8
011 ₁₀	0B ₁₆	(vt)	034 ₁₀	22 ₁₆	”	057 ₁₀	39 ₁₆	9
012 ₁₀	0C ₁₆	(np)	035 ₁₀	23 ₁₆	#	058 ₁₀	3A ₁₆	:
013 ₁₀	0D ₁₆	(cr)	036 ₁₀	24 ₁₆	\$	059 ₁₀	3B ₁₆	;
014 ₁₀	0E ₁₆	(so)	037 ₁₀	25 ₁₆	%	060 ₁₀	3C ₁₆	i
015 ₁₀	0F ₁₆	(si)	038 ₁₀	26 ₁₆	&	061 ₁₀	3D ₁₆	=
016 ₁₀	10 ₁₆	(dle)	039 ₁₀	27 ₁₆	,	062 ₁₀	3E ₁₆	¿
017 ₁₀	11 ₁₆	(dc1)	040 ₁₀	28 ₁₆	(063 ₁₀	3F ₁₆	?
018 ₁₀	12 ₁₆	(dc2)	041 ₁₀	29 ₁₆)	064 ₁₀	40 ₁₆	@
019 ₁₀	13 ₁₆	(dc3)	042 ₁₀	2A ₁₆	*	065 ₁₀	41 ₁₆	A
020 ₁₀	14 ₁₆	(dc4)	043 ₁₀	2B ₁₆	+	066 ₁₀	42 ₁₆	B
021 ₁₀	15 ₁₆	(nak)	044 ₁₀	2C ₁₆	,	067 ₁₀	43 ₁₆	C
022 ₁₀	16 ₁₆	(syn)	045 ₁₀	2D ₁₆	-	068 ₁₀	44 ₁₆	D

0x45 a 0x7F:

069 ₁₀	45 ₁₆	E	092 ₁₀	5C ₁₆	\	115 ₁₀	73 ₁₆	s
070 ₁₀	46 ₁₆	F	093 ₁₀	5D ₁₆]	116 ₁₀	74 ₁₆	t
071 ₁₀	47 ₁₆	G	094 ₁₀	5E ₁₆	^	117 ₁₀	75 ₁₆	u
072 ₁₀	48 ₁₆	H	095 ₁₀	5F ₁₆	-	118 ₁₀	76 ₁₆	v
073 ₁₀	49 ₁₆	I	096 ₁₀	60 ₁₆	‘	119 ₁₀	77 ₁₆	w
074 ₁₀	4A ₁₆	J	097 ₁₀	61 ₁₆	a	120 ₁₀	78 ₁₆	x
075 ₁₀	4B ₁₆	K	098 ₁₀	62 ₁₆	b	121 ₁₀	79 ₁₆	y
076 ₁₀	4C ₁₆	L	099 ₁₀	63 ₁₆	c	122 ₁₀	7A ₁₆	z
077 ₁₀	4D ₁₆	M	100 ₁₀	64 ₁₆	d	123 ₁₀	7B ₁₆	{
078 ₁₀	4E ₁₆	N	101 ₁₀	65 ₁₆	e	124 ₁₀	7C ₁₆	
079 ₁₀	4F ₁₆	O	102 ₁₀	66 ₁₆	f	125 ₁₀	7D ₁₆	}
080 ₁₀	50 ₁₆	P	103 ₁₀	67 ₁₆	g	126 ₁₀	7E ₁₆	~
081 ₁₀	51 ₁₆	Q	104 ₁₀	68 ₁₆	h	127 ₁₀	7F ₁₆	DEL
082 ₁₀	52 ₁₆	R	105 ₁₀	69 ₁₆	i			
083 ₁₀	53 ₁₆	S	106 ₁₀	6A ₁₆	j			
084 ₁₀	54 ₁₆	T	107 ₁₀	6B ₁₆	k			
085 ₁₀	55 ₁₆	U	108 ₁₀	6C ₁₆	l			
086 ₁₀	56 ₁₆	V	109 ₁₀	6D ₁₆	m			
087 ₁₀	57 ₁₆	W	110 ₁₀	6E ₁₆	n			
088 ₁₀	58 ₁₆	X	111 ₁₀	6F ₁₆	o			
089 ₁₀	59 ₁₆	Y	112 ₁₀	70 ₁₆	p			
090 ₁₀	5A ₁₆	Z	113 ₁₀	71 ₁₆	q			
091 ₁₀	5B ₁₆	[114 ₁₀	72 ₁₆	r			

Apêndice B

Fluxograma dos registradores de tempo

A figura a seguir complementa o conteúdo do Capítulo 5 e mostra como se comportam alguns dos registradores de tempo do ARM, se não houver nenhuma interrupção, por exemplo, uma chamada que force a contagem a zero.

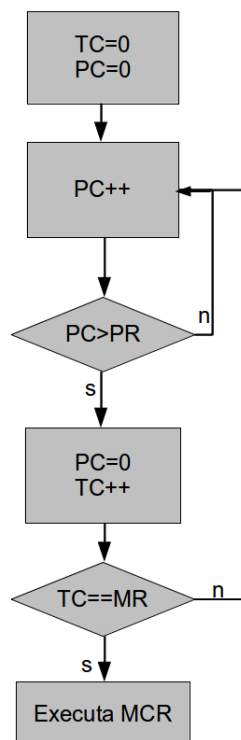


Figura B.1: Fluxograma indicando os registradores de tempo